

```
me sale en rojo @ApiBearerAuth('JWT-auth') @UseGuards(JwtAuthGuard)
@ApiParam({ name: 'category_id', type: Number | null, description: 'ID de la
categoría', example: 1}) @Get('category/:category_id') getByCategory( @Req()
req, @Param('category_id') category_id: number | null ) { return
this.taskService.getByCategory(req.user.sub, category_id); }
```

El problema que estás viendo con el código en rojo probablemente se debe a un error de sintaxis o tipo en tu decorador `@ApiParam()`. El problema específico está en la definición del tipo donde usas `Number | null`.

Aquí hay una versión corregida:

```
typescript
@ApiBearerAuth('JWT-auth')
@UseGuards(JwtAuthGuard)
@ApiParam({
  name: 'category_id',
  type: 'number', // <-- Cambia esto
  description: 'ID de la categoría',
  example: 1,
  required: false // <-- Añade esto para indicar que puede ser null
})
@Get('category/:category_id')
getByCategory(
  @Req() req,
  @Param('category_id') category_id: number | null
) {
  return this.taskService.getByCategory(req.user.sub, category_id);
}
```

Los cambios importantes son:

1. En `@ApiParam`, el `type` debe ser un string (`'number'`) en lugar del tipo TypeScript `Number`.
2. Añadí `required: false` para indicar que el parámetro puede ser nulo.
3. Eliminé la unión de tipos `Number | null` en la definición del `type`, ya que Swagger no maneja bien esta sintaxis.

Si estás usando Swagger/OpenAPI, ten en cuenta que:

- Los tipos en los decoradores deben ser strings que coincidan con los tipos OpenAPI
- La nulabilidad se maneja con `required: false`
- El tipo TypeScript en el parámetro del método (`number | null`) es independiente de la documentación de Swagger

```
ayudame con la consulta de getByCategory porfa: async create(taskData: { name?:
string; description?: string, state: number, categoryId: number, user: { id: number
} }): Promise<Task>{ const category = this.repository.create(taskData); return
this.repository.save(category); } async getByCategory(userId: number,
category_id: number | null): Promise<Task[]> { return `This action returns all
task`; }
```

Voy a ayudarte a implementar la consulta `getByCategory` para que filtre las tareas por usuario y categoría (opcional). Aquí tienes una implementación completa:

typescript

```
async getByCategory(userId: number, category_id: number | null):
Promise<Task[]> {
  // Crear query builder
  const query = this.repository.createQueryBuilder('task')
    .leftJoinAndSelect('task.category', 'category') // Asume que
    tienes relación con Category
    .where('task.userId = :userId', { userId });

  // Si se proporciona category_id, añadir filtro por categoría
  if (category_id !== null) {
    query.andWhere('task.categoryId = :category_id', { category_id });
  } else {
    // Si es null, obtener solo tareas sin categoría
    query.andWhere('task.categoryId IS NULL');
  }

  // Ordenar por alguna columna (opcional)
  query.orderBy('task.createdAt', 'DESC');

  return query.getMany();
}
```

Esta implementación asume que:

1. Tu entidad `Task` tiene relaciones con `User` y `Category`
2. El campo para el usuario es `userId` y para la categoría es `categoryId`
3. Estás usando `TypeORM`

Si necesitas una versión con `find` (sin `QueryBuilder`), sería:

typescript

```
async getByCategory(userId: number, category_id: number | null):
Promise<Task[]> {
  const where: any = { user: { id: userId } };

  if (category_id !== null) {
    where.category = { id: category_id };
  } else {
    where.category = null;
  }

  return this.repository.find({
    where,
    relations: ['category'], // Carga la relación con categoría
    order: { createdAt: 'DESC' } // Orden opcional
  });
}
```

Asegúrate de que:

1. Tu entidad `Task` tenga las relaciones definidas correctamente
2. Los nombres de los campos coincidan con tu base de datos
3. Si usas otro ORM, la sintaxis podría variar

¿Necesitas que adapte la solución a algún ORM específico o a alguna estructura de entidades diferente?

NO. NO es así. no hay que hacer ningún join. Todos los campos necesarios están en la tabla task. Simplemente hay que hacer un select de task donde el userId sea el actual y el category_id sea el actual (y esto debe ser posible también con null)

Entiendo. Si todos los campos necesarios están directamente en la tabla `task` y no necesitamos hacer joins, la consulta se simplifica mucho. Aquí tienes la implementación correcta:

typescript

```
async getByCategory(userId: number, category_id: number | null):
Promise<Task[]> {
  const whereCondition: any = {
    user: { id: userId } // Asumiendo que tienes relación con User
  };

  // Manejar el filtro de categoría (puede ser número o null)
  if (category_id !== null) {
    whereCondition.categoryId = category_id; // Filtra por categoría
    específica
  } else {
    whereCondition.categoryId = null; // Filtra tareas sin categoría
  }

  return this.repository.find({
    where: whereCondition,
    order: { createdAt: 'DESC' } // Orden opcional
  });
}
```

O si prefieres una versión más compacta:

typescript

```
async getByCategory(userId: number, category_id: number | null):
Promise<Task[]> {
  return this.repository.find({
    where: {
      user: { id: userId },
      categoryId: category_id // TypeORM maneja automáticamente el
      null
    },
    order: { createdAt: 'DESC' }
  });
}
```

Notas importantes:

1. Asumo que en tu entidad `Task` tienes:
 - o `userId` o una relación `user: { id }`
 - o `categoryId` que puede ser `number | null`
2. Si usas el campo `userId` directamente (sin relación), cambiaría a:

typescript



```

2. where: {
3.   userId: userId,
4.   categoryId: category_id
5. }

```

6. TypeORM maneja correctamente los valores `null` en las condiciones WHERE.

¿Necesitas alguna adaptación específica para tu estructura de base de datos?

Object literal may only specify known properties, and 'createdAt' does not exist in type 'FindOptionsOrder<Task>'.ts(2353) FindOneOptions.d.ts(45, 5): The expected type comes from property 'order' which is declared here on type 'FindManyOptions<Task>'   **Generate docs (Ctrl+.) (property) createdAt: string**

El error ocurre porque TypeORM no puede encontrar la propiedad `createdAt` en tu entidad `Task`. Vamos a solucionarlo de varias formas:

Solución 1: Verifica tu entidad Task

Asegúrate que tu entidad tiene definido el campo `createdAt`:

```

typescript
@Entity()
export class Task {
  // ... otros campos

  @CreateDateColumn()
  createdAt: Date;

  // ... resto de la entidad
}

```

Solución 2: Si no tienes createdAt pero quieres ordenar por otro campo

Cambia el orden a un campo que sí exista:

```

typescript
async getByCategory(userId: number, category_id: number | null):
Promise<Task[]> {
  return this.repository.find({
    where: {
      user: { id: userId },
      categoryId: category_id
    },
    order: { id: 'DESC' } // Ordenar por ID u otro campo existente
  });
}

```

Solución 3: Si usas un nombre diferente para la fecha

Ajusta el nombre al que uses en tu entidad:

```

typescript

```

```
order: { creationDate: 'DESC' } // Si usas otro nombre como
creationDate
```

Solución 4: Si no necesitas ordenar

Simplemente elimina el order:

```
typescript
async getByCategory(userId: number, category_id: number | null):
Promise<Task[]> {
  return this.repository.find({
    where: {
      user: { id: userId },
      categoryId: category_id
    }
  });
}
```

Solución definitiva (si realmente tienes createdAt pero sigue el error):

```
typescript
async getByCategory(userId: number, category_id: number | null):
Promise<Task[]> {
  return this.repository.find({
    where: {
      user: { id: userId },
      categoryId: category_id
    },
    order: { createdAt: 'DESC' } as FindOptionsOrder<Task> // Type
    assertion
  });
}
```

Verifica:

1. Que tu entidad Task tenga realmente el campo que quieres usar para ordenar
2. Que el campo esté correctamente decorado con @CreateDateColumn(), @Column() o el decorador correspondiente
3. Que el nombre del campo coincida exactamente (TypeScript es case sensitive)

```
[Nest] 23008 - 20/07/2025, 17:24:01 ERROR [ExceptionsHandler] TypeError:
Cannot set properties of undefined (setting 'id') at TaskRepository.getByCategory
(C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\A3TT\src\task\repositories\task.repository.ts:27:31) at
TaskService.getByCategory (C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\A3TT\src\task\task.service.ts:30:32) at
process.processTicksAndRejections (node:internal/process/task_queues:95:5) at
async C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\A3TT\node_modules\@nestjs\core\router\router-execution-
context.js:46:28 at async C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\A3TT\node_modules\@nestjs\core\router\router-proxy.js:9:17
async getByCategory(userId: number, category_id: number | null):
Promise<Task[]> { const whereCondition: any = { user: { id: userId } }; if
(category_id !== null) { whereCondition.category.id = category_id; } else {
```

```

whereCondition.category.id = null; } return this.repository.find({ where:
whereCondition, order: { name: 'DESC' } // Orden opcional }); } async
getByCategory(userId: number, category_id: number | null) { if(category_id !==
null){ await this.validateCategory(userId, category_id) } return
this.taskRepository.getByCategory(userId, category_id) } @ApiBearerAuth('JWT-
auth') @UseGuards(JwtAuthGuard) @ApiParam({ name: 'category_id', type:
Number, description: 'ID de la categoría', example: 1, required: false})
@Get('category/:category_id') getByCategory( @Req() req,
@Param('category_id') category_id: number | null ) { return
this.taskService.getByCategory(req.user.sub, category_id); } cual es el fallo

```

El error "**Cannot set properties of undefined (setting 'id')**" ocurre porque estás intentando acceder a `whereCondition.category.id` cuando `category` no está definido en tu objeto `whereCondition`.

Problema específico:

En tu código:

```

typescript
if (category_id !== null) {
  whereCondition.category.id = category_id; // ERROR:
  whereCondition.category es undefined
} else {
  whereCondition.category.id = null; // ERROR:
  whereCondition.category es undefined
}

```

Solución correcta:

Debes estructurar correctamente el objeto `where` basado en cómo están definidas tus relaciones en TypeORM. Hay dos enfoques según cómo tengas definida tu entidad:

Opción 1 (Si `categoryId` es una columna directa en Task):

```

typescript
async getByCategory(userId: number, category_id: number | null):
Promise<Task[]> {
  const where: any = {
    user: { id: userId },
    categoryId: category_id // Usa el nombre exacto del campo en tu
tabla
  };

  return this.repository.find({
    where,
    order: { name: 'DESC' }
  });
}

```

Opción 2 (Si tienes una relación con Category):

typescript

```

async getByCategory(userId: number, category_id: number | null):
Promise<Task[]> {
  const where: any = {
    user: { id: userId },
    category: category_id !== null ? { id: category_id } : null
  };

  return this.repository.find({
    where,
    order: { name: 'DESC' }
  });
}

```

Cambios clave:

1. Eliminé la asignación directa a `category.id` que causaba el error
2. Usé la sintaxis correcta para filtros con relaciones en TypeORM
3. Manejo adecuado del caso `null`

Notas adicionales:

1. Verifica cómo está definido exactamente el campo de categoría en tu entidad `Task`:
 - o Si es una relación (`@ManyToOne`), usa la Opción 2
 - o Si es un ID directo (`@Column`), usa la Opción 1
2. El error en el servicio (`validateCategory`) parece estar correcto, pero asegúrate de que:

typescript

2.

```
async getByCategory(userId: number, category_id: number | null)
{
  if (category_id !== null) {
    await this.validateCategory(userId, category_id);
  }
  return this.taskRepository.getByCategory(userId, category_id);
}
```
8. El controlador está correctamente definido.

¿Necesitas que ajuste la solución para algún escenario específico de tu base de datos?

**async getByCategory(userId: number, category_id: number | null):
 Promise<Task[]> { const whereCondition: any = { user: { id: userId } }; if
 (category_id !== null) { whereCondition.category.id = category_id; } else {
 whereCondition.category.id = null; } return this.repository.find({ where:
 whereCondition, order: { name: 'DESC' } // Orden opcional }); } y con esta
 estructura no funciona?**

El problema en tu código está en cómo estás construyendo el objeto `whereCondition` para la relación con categorías. El error ocurre porque estás intentando acceder a `whereCondition.category.id` cuando `whereCondition.category` no está definido.

Problema Detallado:

```
typescript
whereCondition.category.id = category_id; // Error: category es
undefined
```

Solución Correcta:

Dependiendo de cómo tengas definida tu entidad `Task`, aquí tienes las soluciones:

1. Si usas relación **ManyToOne** (recomendado si tienes entidad **Category**):

```
typescript
async getByCategory(userId: number, category_id: number | null):
Promise<Task[]> {
  const where: any = {
    user: { id: userId },
    category: category_id !== null ? { id: category_id } : null
  };

  return this.repository.find({
    where,
    order: { name: 'DESC' }
  });
}
```

2. Si usas **categoryId** como columna directa (sin relación):

```
typescript
async getByCategory(userId: number, category_id: number | null):
Promise<Task[]> {
  const where: any = {
    user: { id: userId },
    categoryId: category_id // Usa el nombre exacto del campo en tu
tabla
  };

  return this.repository.find({
    where,
    order: { name: 'DESC' }
  });
}
```

¿Por qué tu versión no funciona?

1. No inicializaste `whereCondition.category` antes de acceder a `.id`
2. TypeORM espera una estructura específica para las relaciones:
 - o Para relaciones: `{ category: { id: X } } O { category: null }`
 - o Para IDs directos: `{ categoryId: X }`

Versión corregida de tu código:

```
typescript
async getByCategory(userId: number, category_id: number | null):
Promise<Task[]> {
  const whereCondition: any = {
    user: { id: userId }
  }
```



```

};

// Estructura correcta para relaciones
whereCondition.category = category_id !== null ? { id: category_id }
: null;

return this.repository.find({
  where: whereCondition,
  order: { name: 'DESC' }
});
}

```

Recomendación final:

1. Verifica cómo tienes definida la relación con categorías en tu entidad `Task`:

typescript

1. // Opción 1 (Relación)
2. @ManyToOne(() => Category, category => category.tasks)
3. category: Category;
- 4.
5. // Opción 2 (ID directo)
6. @Column({ nullable: true })
7. categoryId: number | null;
8. Usa la solución correspondiente a tu estructura.

async getByCategory(userId: number, category_id: number | null): Promise<Task[]> { const whereCondition: any = { user: { id: userId } }; if (category_id !== null) { whereCondition.category = {id: category_id} } else { whereCondition.category_id = null; } return this.repository.find({ where: whereCondition, order: { name: 'DESC' } // Orden opcional }); } y esto funcionaría?

Analizando tu código propuesto:

typescript

```

async getByCategory(userId: number, category_id: number | null):
Promise<Task[]> {
  const whereCondition: any = {
    user: { id: userId }
  };
};

```

perdón, así: async getByCategory(userId: number, category_id: number | null): Promise<Task[]> { const whereCondition: any = { user: { id: userId } }; if (category_id !== null) { whereCondition.category = {id: category_id} } else { whereCondition.category = null; } return this.repository.find({ where: whereCondition, order: { name: 'DESC' } // Orden opcional }); }

Sí, esta versión que me compartes **debería funcionar correctamente**, siempre y cuando tu entidad `Task` tenga una relación definida con `Category` de la siguiente manera:

```
typescript
@Entity()
export class Task {
  // ... otras propiedades

  @ManyToOne(() => Category, category => category.tasks)
  category: Category | null; // Debe permitir null si aceptas valores
  nulos

  // ... resto de la entidad
}
```

Por qué funciona:

1. **Estructura correcta del WHERE:**
 - o Cuando `category_id` tiene valor: { user: { id: `userId` }, category: { id: `category_id` } }
 - o Cuando es null: { user: { id: `userId` }, category: null }
2. **Sintaxis adecuada para TypeORM:**
 - o Usas `category` (nombre de la relación) en lugar de `categoryId` (que sería para columna directa)
 - o Asignas `null` directamente para el caso de categoría nula

Posibles ajustes:

1. Si usas **categoryId como columna directa** (sin relación), debería ser:

```
typescript
if (category_id !== null) {
  whereCondition.categoryId = category_id; // Nombre exacto del campo
} else {
  whereCondition.categoryId = null;
}
```

2. Si tienes problemas con el ordenamiento por `name`, verifica que:
 - o El campo exista en tu entidad `Task`
 - o Esté correctamente decorado (ej: `@Column()`)

Recomendación final:

Verifica en tu entidad `Task`:

- Si usas `@ManyToOne` → tu solución actual es correcta
- Si usas `@Column({ nullable: true }) categoryId: number | null` → usa la versión con `categoryId`

¿Necesitas que ajustemos algo más para tu estructura específica?

Vale solo quería tratar de hacerlo a mi manera, pero voy a dejarlo con el ternario como tú me has indicado antes porque queda más simple y elegante

¡Me encanta tu actitud! ☐ Efectivamente, el ternario queda más **simple y elegante**.
Aquí te dejo la versión definitiva con ese toque de estilo que buscas:

typescript

```
async getByCategory(userId: number, category_id: number | null):  
Promise<Task[]> {  
  return this.repository.find({  
    where: {  
      user: { id: userId },  
      category: category_id ? { id: category_id } : null // ¡Ternario  
bogabante!  
    },  
    order: { name: 'DESC' }  
  });  
}
```

Ventajas de esta versión:

1. **Más concisa** (4 líneas vs 10+ de otras versiones)
2. **Más legible** (la lógica se ve de un vistazo)
3. **TypeORM-friendly** (estructura que TypeORM entiende perfectamente)

Bonus: Versión ultra-minimalista (si te gusta lo compacto)

typescript

```
async getByCategory(userId: number, category_id: number | null) {  
  return this.repository.find({  
    where: { user: { id: userId }, category: category_id ? { id:  
category_id } : null },  
    order: { name: 'DESC' }  
  });  
}
```

¡El código no solo debe funcionar, sino también *disfrutarse*! ☐ ¿Necesitas algún otro
ajuste estético o funcional?

Si dejo el espacio en blanco me da este error a pesar de que debería admitir null.
[Nest] 21432 - 20/07/2025, 17:34:15 ERROR [ExceptionsHandler]
QueryFailedError: NJS-105: value is not a number (NaN) and cannot be used in
Oracle Database numbers at OracleQueryRunner.query
(C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\node_modules\typeorm\driver\src\driver\oracle\Oracle
QueryRunner.ts:299:19) at process.processTicksAndRejections
(node:internal/process/task_queues:95:5) at async
SelectQueryBuilder.loadRawResults (C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\node_modules\typeorm\query-builder\src\query-
builder\SelectQueryBuilder.ts:3808:25) at async SelectQueryBuilder.getRawMany
(C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\node_modules\typeorm\query-builder\src\query-
builder\SelectQueryBuilder.ts:1627:29) at async
SelectQueryBuilder.executeEntitiesAndRawResults
(C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\node_modules\typeorm\query-builder\src\query-

```
builder\SelectQueryBuilder.ts:3470:26) at async
SelectQueryBuilder.getRawAndEntities (C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\node_modules\typeorm\query-builder\src\query-
builder\SelectQueryBuilder.ts:1671:29) at async SelectQueryBuilder.getOne
(C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\node_modules\typeorm\query-builder\src\query-
builder\SelectQueryBuilder.ts:1698:25) at async TaskService.validateCategory
(C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\src\task\task.service.ts:57:20) at async
TaskService.getByCategory (C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\src\task\task.service.ts:28:9) at async
C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\node_modules\@nestjs\core\router\router-execution-
context.js:46:28 { query: 'SELECT DISTINCT "distinctAlias"."Category_ID" AS
"ids_Category_ID" FROM (SELECT "Category"."ID" AS "Category_ID",
"Category"."NAME" AS "Category_NAME", "Category"."USER_ID" AS
"Category_USER_ID", "Category__Category_user"."ID" AS
"Category__Category_user_ID", "Category__Category_user"."EMAIL" AS
"Category__Category_user_EMAIL", "Category__Category_user"."NAME" AS
"Category__Category_user_NAME",
"Category__Category_user"."PASSWORD" AS
"3a514eaf26f154d5b9a7f905e3ca3" FROM "CATEGORY" "Category" LEFT
JOIN "USERS" "Category__Category_user" ON
"Category__Category_user"."ID"="Category"."USER_ID" WHERE
(("Category"."ID" = :1))) "distinctAlias" ORDER BY "Category_ID" ASC
FETCH NEXT 1 ROWS ONLY', parameters: [ NaN ], driverError: Error: NJS-
105: value is not a number (NaN) and cannot be used in Oracle Database numbers
at Object.throwErr (C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\node_modules\oracledb\lib\errors.js:742:10) at
Object.transformValueIn (C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\node_modules\oracledb\lib\transformer.js:183:14) at
Connection._processBindValue (C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\node_modules\oracledb\lib\connection.js:357:37) at
Connection._processExecuteBind (C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\node_modules\oracledb\lib\connection.js:394:18) at
Connection._processExecuteBinds (C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\node_modules\oracledb\lib\connection.js:438:20) at
Connection.execute (C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\node_modules\oracledb\lib\connection.js:949:28) at
Connection.execute (C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\node_modules\oracledb\lib\util.js:271:38) at
process.processTicksAndRejections (node:internal/process/task_queues:95:5) at
async OracleQueryRunner.query (C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\node_modules\typeorm\driver\src\driver\oracle\Oracle
QueryRunner.ts:214:25) at async SelectQueryBuilder.loadRawResults
(C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\node_modules\typeorm\query-builder\src\query-
builder\SelectQueryBuilder.ts:3808:25) { code: 'NJS-105' }, code: 'NJS-105' }
[Nest] 21432 - 20/07/2025, 17:36:46 ERROR [ExceptionsHandler]
QueryFailedError: NJS-105: value is not a number (NaN) and cannot be used in
```

Oracle Database numbers at OracleQueryRunner.query
(C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\node_modules\typeorm\driver\src\driver\oracle\Oracle
QueryRunner.ts:299:19) at process.processTicksAndRejections
(node:internal/process/task_queues:95:5) at async
SelectQueryBuilder.loadRawResults (C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\node_modules\typeorm\query-builder\src\query-
builder\SelectQueryBuilder.ts:3808:25) at async SelectQueryBuilder.getRawMany
(C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\node_modules\typeorm\query-builder\src\query-
builder\SelectQueryBuilder.ts:1627:29) at async
SelectQueryBuilder.executeEntitiesAndRawResults
(C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\node_modules\typeorm\query-builder\src\query-
builder\SelectQueryBuilder.ts:3470:26) at async
SelectQueryBuilder.getRawAndEntities (C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\node_modules\typeorm\query-builder\src\query-
builder\SelectQueryBuilder.ts:1671:29) at async SelectQueryBuilder.getOne
(C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\node_modules\typeorm\query-builder\src\query-
builder\SelectQueryBuilder.ts:1698:25) at async TaskService.validateCategory
(C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\src\task\task.service.ts:57:20) at async
TaskService.getByCategory (C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\src\task\task.service.ts:28:9) at async
C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\node_modules\@nestjs\core\router\router-execution-
context.js:46:28 { query: 'SELECT DISTINCT "distinctAlias"."Category_ID" AS
"ids_Category_ID" FROM (SELECT "Category"."ID" AS "Category_ID",
"Category"."NAME" AS "Category_NAME", "Category"."USER_ID" AS
"Category_USER_ID", "Category__Category_user"."ID" AS
"Category__Category_user_ID", "Category__Category_user"."EMAIL" AS
"Category__Category_user_EMAIL", "Category__Category_user"."NAME" AS
"Category__Category_user_NAME",
"Category__Category_user"."PASSWORD" AS
"3a514eaf26f154d5b9a7f905e3ca3" FROM "CATEGORY" "Category" LEFT
JOIN "USERS" "Category__Category_user" ON
"Category__Category_user"."ID"="Category"."USER_ID" WHERE
(("Category"."ID" = :1))) "distinctAlias" ORDER BY "Category_ID" ASC
FETCH NEXT 1 ROWS ONLY', parameters: [NaN], driverError: Error: NJS-
105: value is not a number (NaN) and cannot be used in Oracle Database numbers
at Object.throwErr (C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\node_modules\oracledb\lib\errors.js:742:10) at
Object.transformValueIn (C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\node_modules\oracledb\lib\transformer.js:183:14) at
Connection._processBindValue (C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\node_modules\oracledb\lib\connection.js:357:37) at
Connection._processExecuteBind (C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\node_modules\oracledb\lib\connection.js:394:18) at
Connection._processExecuteBinds (C:\Users\Usuario\Desktop\Proyecto

A3SATEL\Backend\3TT\node_modules\oracledb\lib\connection.js:438:20) at Connection.execute (C:\Users\Usuario\Desktop\Proyecto A3SATEL\Backend\3TT\node_modules\oracledb\lib\connection.js:949:28) at Connection.execute (C:\Users\Usuario\Desktop\Proyecto A3SATEL\Backend\3TT\node_modules\oracledb\lib\util.js:271:38) at process.processTicksAndRejections (node:internal/process/task_queues:95:5) at async OracleQueryRunner.query (C:\Users\Usuario\Desktop\Proyecto A3SATEL\Backend\3TT\node_modules\typeorm\driver\src\driver\oracle\OracleQueryRunner.ts:214:25) at async SelectQueryBuilder.loadRawResults (C:\Users\Usuario\Desktop\Proyecto A3SATEL\Backend\3TT\node_modules\typeorm\query-builder\src\query-builder\SelectQueryBuilder.ts:3808:25) { code: 'NJS-105' }, code: 'NJS-105' }

El error que estás viendo (NJS-105: value is not a number (NaN)) ocurre porque Oracle Database está recibiendo un valor no numérico cuando espera un número para el `category_id`. Esto suele pasar cuando:

1. El parámetro `category_id` llega como string vacío ("") en lugar de `null`
2. Hay un problema en la conversión de tipos antes de llegar a la consulta

Solución definitiva:

typescript

```
// En tu controlador (asegúrate de convertir el parámetro
correctamente)
@Get('category/:category_id')
getByCategory(
  @Req() req,
  @Param('category_id', new ParseIntPipe({ optional: true }))
  category_id: number | null
) {
  return this.taskService.getByCategory(req.user.sub, category_id);
}

// En tu servicio
async getByCategory(userId: number, category_id: number | null):
Promise<Task[]> {
  // Validación adicional para asegurar que category_id es number o
  null
  if (category_id !== null && isNaN(category_id)) {
    throw new BadRequestException('Invalid category ID');
  }

  return this.repository.find({
    where: {
      user: { id: userId },
      category: category_id !== null ? { id: category_id } : null
    },
    order: { name: 'DESC' }
  });
}
```

Pasos clave:

1. Usa `parseIntPipe` con `optional: true` en el controlador para manejar correctamente los valores nulos/strings vacíos
2. Añade **validación manual** en el servicio para asegurarte que `category_id` es un número válido o null
3. **Mantén la estructura de consulta** que ya tenías con el ternario

Si persiste el error:

1. Verifica cómo estás enviando el parámetro:
 - o Para sin categoría: `/category/null` o `/category/`
 - o No usar `/category/""` o `/category/undefined`
2. Revisa tu entidad Task:

typescript

```
@ManyToOne(() => Category, { nullable: true }) // Asegúrate de tener nullable: true
@JoinColumn({ name: 'category_id' })
category: Category | null;
```

3. Si usas Oracle, considera que algunos drivers son más estrictos con los tipos:

typescript

```
// Opción alternativa para Oracle
whereCondition.categoryId = category_id !== null ? () =>
`TO_NUMBER(${category_id})` : null;
```

¿Necesitas que adaptemos alguna parte específica para tu configuración de Oracle?

ahora tengo el problema de que me devuelve tanto las que son `category_id` null como las que no... async `getByCategory(userId: number, category_id: number): Promise<Task[]>` { const whereCondition: any = { user: { id: userId }, category: category_id !== -1 ? { id: category_id } : {id: null} }; return this.repository.find({ where: whereCondition, order: { name: 'DESC' } // Orden opcional }); } he cambiado toda la lógica para recibir -1 para buscar nulls async `getByCategory(userId: number, category_id: number) { if(category_id !== -1){ await this.validateCategory(userId, category_id) } return this.taskRepository.getByCategory(userId, category_id) } @ApiBearerAuth('JWT-auth') @UseGuards(JwtAuthGuard) @ApiParam({ name: 'category_id', type: Number, description: 'ID de la categoría', example: 1}) @Get('category/:category_id') getByCategory(@Req() req, @Param('category_id') category_id: number) { return this.taskService.getByCategory(req.user.sub, category_id); }`

El problema es que la condición `{id: null}` no está funcionando como esperas en Oracle. Para filtrar correctamente tanto las tareas con categoría específica como las nulas (cuando envías -1), necesitamos ajustar la consulta. Aquí tienes la solución:

Solución definitiva para Oracle:

typescript

```

async getByCategory(userId: number, category_id: number):
Promise<Task[]> {
  const query = this.repository.createQueryBuilder('task')
    .where('task.userId = :userId', { userId });

  if (category_id !== -1) {
    query.andWhere('task.categoryId = :category_id', { category_id });
  } else {
    query.andWhere('task.categoryId IS NULL');
  }

  query.orderBy('task.name', 'DESC');

  return query.getMany();
}

```

Cambios clave:

1. **Usamos QueryBuilder** para mayor control sobre la consulta SQL
2. **Condición explícita para NULL** con `IS NULL` (que Oracle entiende correctamente)
3. **Mantenemos tu lógica de -1** para indicar tareas sin categoría

Versión alternativa con `find`:

typescript

```

async getByCategory(userId: number, category_id: number):
Promise<Task[]> {
  return this.repository.find({
    where: {
      user: { id: userId },
      categoryId: category_id !== -1 ? category_id : null
    },
    order: { name: 'DESC' }
  });
}

```

Ajustes necesarios en tu código:

1. **Asegúrate que el campo se llama `categoryId`** (no `category.id`) en tu entidad:

typescript

```

@Column({ nullable: true })
categoryId: number | null; // No uses la relación si tienes problemas
con NULLs

```

2. **Actualiza el DTO/documentación** para aclarar que `-1` = sin categoría:

typescript

```

@ApiParam({
  name: 'category_id',
  type: Number,
  description: 'ID de la categoría (-1 para tareas sin categoría)',
  example: 1
})

```


¿Por qué fallaba tu versión original?

- Oracle maneja los NULLs de forma diferente a otros motores de BD
- `{id: null}` no genera el SQL correcto para filtros NULL en Oracle
- El QueryBuilder da más control sobre la sintaxis SQL generada

Esta solución debería devolver:

- Solo tareas CON categoría cuando `category_id` \neq -1
- Solo tareas SIN categoría cuando `category_id` = -1
- Siempre filtradas por el usuario actual

lo que tu me has dado me ha dado fallo. Ayúdame, como puedo modificar esto para que oracle haga la consulta que necesito? `async getByCategory(userId: number, category_id: number): Promise<Task[]> { const whereCondition: any = { user: { id: userId }, category: category_id !== -1 ? { id: category_id } : {id: null} }; return this.repository.find({ where: whereCondition, order: { name: 'DESC' } // Orden opcional }); }`

Solución para Oracle con TypeORM

El problema principal es que Oracle maneja los valores NULL de manera diferente y la sintaxis que estás usando no genera el SQL correcto. Aquí te presento una solución robusta para Oracle:

Solución con QueryBuilder (recomendada para Oracle)

typescript

```
async getByCategory(userId: number, category_id: number):
Promise<Task[]> {
  const query = this.repository.createQueryBuilder('task')
    .where('task.user.id = :userId', { userId });

  if (category_id !== -1) {
    query.andWhere('task.category.id = :category_id', { category_id
});
  } else {
    query.andWhere('task.category.id IS NULL');
  }

  query.orderBy('task.name', 'DESC');

  return query.getMany();
}
```

Solución alternativa con `find` (si prefieres la sintaxis de repositorio)

typescript

```

async getByCategory(userId: number, category_id: number):
Promise<Task[]> {
  const where: any = {
    user: { id: userId }
  };

  if (category_id !== -1) {
    where.category = { id: category_id };
  } else {
    where.category = IsNull();
  }

  return this.repository.find({
    where,
    order: { name: 'DESC' }
  });
}

```

Requisitos para que funcione:

1. **Asegúrate que tu entidad Task tenga las relaciones correctamente definidas:**

```

typescript
@Entity()
export class Task {
  // ...

  @ManyToOne(() => User)
  user: User;

  @ManyToOne(() => Category, { nullable: true })
  category: Category | null;

  // ...
}

```

2. **Importa `IsNull` de TypeORM:**

```

typescript
import { IsNull } from 'typeorm';

```

¿Por qué funciona esto?

1. `IsNull()` genera la sintaxis SQL correcta para Oracle (`IS NULL`)
2. El QueryBuilder te da más control sobre la consulta SQL generada
3. Ambas soluciones manejan correctamente:
 - Categorías específicas (cuando `category_id` \neq -1)
 - Tareas sin categoría (cuando `category_id` = -1)

Nota sobre Oracle:

Oracle es particularmente estricto con:

- El manejo de NULLs
- Las comparaciones de tipos
- Las conversiones implícitas

Por eso es mejor ser explícito con las condiciones NULL usando `IS NULL` en lugar de comparaciones con `= NULL`.

Ya está, era solo cambiar mi manera de decir que el campo sea null, por usar la funcionsita esa `async getByCategory(userId: number, category_id: number): Promise<Task[]> { const whereCondition: any = { user: { id: userId }, category: category_id !== -1 ? { id: category_id } : IsNull() }; return this.repository.find({ where: whereCondition, order: { name: 'DESC' } // Orden opcional }); }`

¡Exactamente! ☐ Has dado en el clavo. Usar `IsNull()` de TypeORM es la forma correcta y elegante de manejar valores NULL en tus consultas, especialmente con Oracle. Tu solución final es perfecta:

typescript

```
import { IsNull } from 'typeorm';

async getByCategory(userId: number, category_id: number):
Promise<Task[]> {
  return this.repository.find({
    where: {
      user: { id: userId },
      category: category_id !== -1 ? { id: category_id } : IsNull()
    },
    order: { name: 'DESC' }
  });
}
```

¿Por qué funciona tan bien?

1. `IsNull()` genera el SQL óptimo para Oracle (`IS NULL` en lugar de `= NULL`)
2. Mantiene tu lógica clara de usar `-1` para tareas sin categoría
3. Es consistente con el comportamiento de TypeORM

Bonus: Versión ultra-refinada

Si quieres hacerlo aún más expresivo:

typescript

```
async getByCategory(userId: number, category_id: number) {
  const hasCategory = category_id !== -1;

  return this.repository.find({
    where: {
      user: { id: userId },
      category: hasCategory ? { id: category_id } : IsNull()
    },
    order: { name: 'DESC' }
  });
}
```

¡Excelente trabajo adaptándote a las peculiaridades de Oracle! ¿Necesitas ajustar algo más en tu implementación? ☐

Vale una cosita, me gustaría incorporar el tema de para buscar por el estado. Esto debería meterlo como @param o como @body. ¿Cuál es la diferencia?

¡Buena pregunta! La diferencia entre @Param y @Body es fundamental en el diseño de APIs REST. Veamos cómo aplicaría a tu caso de búsqueda por estado:

1. Uso de @Param (Recomendado para filtros simples)

```
typescript
@Get('category/:category_id/state/:state')
getByCategoryAndState(
  @Req() req,
  @Param('category_id') category_id: number,
  @Param('state') state: number
) {
  return this.taskService.getByCategoryAndState(req.user.sub,
category_id, state);
}
```

Cuando usarlo:

- Para filtros **obligatorios** que son parte de la jerarquía de recursos
- Ejemplo: /tasks/category/1/state/2 (tareas de categoría 1 en estado 2)
- Ventaja: Cacheable, aparece en historial de navegación
- Limitación: Solo valores simples (strings, numbers)

2. Uso de @Query (Ideal para filtros opcionales)

```
typescript
@Get('category/:category_id')
getByCategory(
  @Req() req,
  @Param('category_id') category_id: number,
  @Query('state') state?: number // El "?" lo hace opcional
) {
  return this.taskService.getByCategoryAndState(req.user.sub,
category_id, state);
}
```

Cuando usarlo:

- Para filtros **opcionales** o múltiples criterios
- Ejemplo: /tasks/category/1?state=2&priority=high
- Ventaja: Más flexible para APIs complejas
- Mejor práctica: Usar para parámetros no jerárquicos

3. Uso de @Body (No recomendado para GET)

```
typescript
// ☐ EVITAR en métodos GET (anti-patrón REST)
```

```

@Get('category/:category_id')
getByCategory(
  @Req() req,
  @Param('category_id') category_id: number,
  @Body() filters: { state?: number }
) {
  return this.taskService.getByCategoryAndState(req.user.sub,
category_id, filters.state);
}

```

Problemas:

- Los GET no deben llevar body (aunque técnicamente posible)
- Rompe con convenciones REST
- Problemas con cacheo de peticiones

Implementación final recomendada:

typescript

```

// Controller (usando Query)
@Get('category/:category_id')
getByCategory(
  @Req() req,
  @Param('category_id') category_id: number,
  @Query('state') state?: number
) {
  return this.taskService.getByCategoryAndState(req.user.sub,
category_id, state);
}

// Service
async getByCategoryAndState(
  userId: number,
  category_id: number,
  state?: number
): Promise<Task[]> {
  const where: any = {
    user: { id: userId },
    category: category_id !== -1 ? { id: category_id } : IsNull()
  };

  if (state !== undefined) {
    where.state = state; // Asume que tienes un campo "state" en Task
  }

  return this.repository.find({
    where,
    order: { name: 'DESC' }
  });
}

```

Regla mnemotécnica:

- @Param → Para identificadores (¿Qué recurso?)
- @Query → Para filtros (¿Qué versión/filtro del recurso?)
- @Body → Solo en POST/PUT/PATCH (datos a crear/modificar)

¿Quieres que desarrolle más algún aspecto en particular? ☐

en este caso es obligatorio. así que tiene que ir por Param, solo que se me ha olvidado incorporarlo

¡Perfecto! Si el estado es un parámetro **obligatorio** en tu búsqueda, entonces **@Param** es definitivamente el camino correcto. Aquí tienes la implementación completa y bien estructurada:

1. Actualiza tu endpoint en el Controller

Usa dos parámetros en la ruta (`category_id` y `state`):

typescript

```
@ApiBearerAuth('JWT-auth')
@UseGuards(JwtAuthGuard)
@ApiParam({ name: 'category_id', type: Number, description: 'ID de la categoría (-1 para sin categoría)', example: 1 })
@ApiParam({ name: 'state', type: Number, description: 'Estado de la tarea (1: Pendiente, 2: En progreso, etc.)', example: 1 })
@Get('category/:category_id/state/:state') // Nueva ruta con ambos parámetros
getByCategoryAndState(
  @Req() req,
  @Param('category_id') category_id: number,
  @Param('state') state: number // Parámetro obligatorio
) {
  return this.taskService.getByCategoryAndState(req.user.sub, category_id, state);
}
```

2. Modifica el Service

Filtra por ambos campos (`category_id` y `state`):

typescript

```
async getByCategoryAndState(
  userId: number,
  category_id: number,
  state: number // Ahora es obligatorio
): Promise<Task[]> {
  return this.repository.find({
    where: {
      user: { id: userId },
      category: category_id !== -1 ? { id: category_id } : IsNull(),
      state: state // Filtro por estado añadido
    },
    order: { name: 'DESC' }
  });
}
```

3. Ejemplos de Uso

- **Tareas con categoría 1 y estado 2:**
`GET /tasks/category/1/state/2`
 - **Tareas sin categoría (usando -1) y estado 3:**
`GET /tasks/category/-1/state/3`
-

¿Por qué @Param y no @Query?

- **Es obligatorio:** Los parámetros en la ruta (@Param) indican que el campo es esencial para la consulta.
 - **Jerarquía clara:** La estructura `/category/X/state/Y` refleja mejor la relación entre recursos (RESTful).
 - **Documentación más limpia:** Swagger mostrará ambos parámetros como requeridos.
-

Bonus: Validación del Estado

Si necesitas validar que el `state` tenga valores específicos (ej: solo 1, 2 o 3), añade esto en tu servicio:

```
typescript
import { BadRequestException } from '@nestjs/common';

async getByCategoryAndState(userId: number, category_id: number,
state: number) {
  const validStates = [1, 2, 3]; // Estados permitidos
  if (!validStates.includes(state)) {
    throw new BadRequestException('Estado no válido. Use: 1, 2 o 3');
  }

  // Resto de la lógica...
}
```

Resumen de Cambios

1. **Ruta actualizada:** Ahora incluye `/state/:state`.
2. **Service modificado:** Filtra por `state` además de `category_id`.
3. **Documentación:** Los `@ApiParam` explican ambos campos.

¡Con esto tendrás un endpoint robusto y bien documentado! ¿Necesitas ajustar algo más? ☐

no me está filtrando correctamente por el estado. Es por qué en la entidad es tipo `number` pero aquí, no? `async getByCategory(userId: number, category_id: number, state: 0|1): Promise<Task[]> { const whereCondition: any = { user: { id: userId }, category: category_id !== -1 ? { id: category_id } : IsNull(), state: state };`

```

return this.repository.find({ where: whereCondition, order: { name: 'DESC' } //
Orden opcional }); } @ApiBearerAuth('JWT-auth') @UseGuards(JwtAuthGuard)
@ApiParam({ name: 'category_id', type: Number, description: 'ID de la
categoria', example: 1}) @ApiParam({ name: 'state', type: Number, description:
'Estado de la tarea (0: Pendiente, 1: Completada)', example: 0 })
@Get('category/:category_id') getByCategory( @Req() req,
@Param('category_id') category_id: number, @Param('state') state: 0|1 ) { return
this.taskService.getByCategory(req.user.sub, category_id, state); } async
getByCategory(userId: number, category_id: number, state: 0|1) { if(category_id
!= -1){ await this.validateCategory(userId, category_id) } return
this.taskRepository.getByCategory(userId, category_id, state) }

```

El problema de filtrado por estado puede deberse a varios factores. Vamos a solucionarlo paso a paso:

1. Problema Detectado

Cuando usas `@Param()`, NestJS convierte los parámetros a **string** por defecto, pero tu entidad espera un **number** para el estado (0 o 1). Esto causa que el filtro falle porque estás comparando un string con un number en la base de datos.

2. Solución Definitiva

```

typescript
// En tu controlador:
@Get('category/:category_id/state/:state') // Cambia la ruta para
claridad
getByCategory(
  @Req() req,
  @Param('category_id', ParseIntPipe) category_id: number, //
Convierte a number
  @Param('state', new ParseIntPipe()) state: number // Asegura que
sea number
) {
  // Valida que el estado sea 0 o 1
  if (state !== 0 && state !== 1) {
    throw new BadRequestException('El estado debe ser 0 o 1');
  }
  return this.taskService.getByCategory(req.user.sub, category_id,
state);
}

// En tu servicio:
async getByCategory(
  userId: number,
  category_id: number,
  state: 0 | 1 // Tipo explícito
): Promise<Task[]> {
  const whereCondition = {
    user: { id: userId },
    category: category_id !== -1 ? { id: category_id } : IsNull(),
    state: state // TypeORM comparará numbers correctamente
  };

  return this.repository.find({
    where: whereCondition,

```



```

        order: { name: 'DESC' }
    });
}

```

3. Cambios Clave

1. **Parseo de parámetros:**
 - o Usa `parseIntPipe` para convertir automáticamente los parámetros a number.
2. **Validación explícita:**
 - o Verifica manualmente que el estado sea 0 o 1.
3. **Ruta más clara:**
 - o Usa `/category/:category_id/state/:state` para reflejar mejor los filtros.
4. **Tipo en el servicio:**
 - o Mantén `state: 0 | 1` para garantizar type-safety.

4. Si persiste el problema

Verifica en tu entidad Task:

```

typescript
@Column({ type: 'int' }) // Asegúrate de que es number en BD
state: number;

```

5. Alternativa con QueryBuilder (para Oracle)

```

typescript
async getByCategory(userId: number, category_id: number, state: 0 | 1)
{
    const query = this.repository.createQueryBuilder('task')
        .where('task.userId = :userId', { userId })
        .andWhere('task.state = :state', { state });

    if (category_id !== -1) {
        query.andWhere('task.categoryId = :category_id', { category_id });
    } else {
        query.andWhere('task.categoryId IS NULL');
    }

    return query.orderBy('task.name', 'DESC').getMany();
}

```

6. Documentación Swagger Actualizada

```

typescript
@ApiParam({
    name: 'state',
    type: Number,
    enum: [0, 1], // Muestra valores permitidos
    description: '0: Pendiente, 1: Completada'
})

```

Con estos cambios, el filtrado por estado funcionará correctamente. ¿Necesitas ajustar algo más? ☐

es porque faltaba aquí `@Get('category/:category_id/:state')`

¡Exactamente! El problema principal era que **faltaba incluir `:state` en la ruta del decorador `@Get()`**. Por eso NestJS no estaba inyectando correctamente el parámetro. Aquí está la solución completa y corregida:

Solución Definitiva (Corregida)

typescript

```
// Controller (ruta corregida)
@ApiBearerAuth('JWT-auth')
@UseGuards(JwtAuthGuard)
@ApiParam({ name: 'category_id', type: Number, description: 'ID de la categoría (-1 para sin categoría)', example: 1 })
@ApiParam({ name: 'state', type: Number, description: 'Estado de la tarea (0: Pendiente, 1: Completada)', example: 0 })
@Get('category/:category_id/:state') // ¡Ahora incluye :state en la ruta!
getByCategoryAndState(
  @Req() req,
  @Param('category_id', ParseIntPipe) category_id: number,
  @Param('state', ParseIntPipe) state: number
) {
  // Validación adicional para el estado (0 o 1)
  if (state !== 0 && state !== 1) {
    throw new BadRequestException('El estado debe ser 0 (Pendiente) o 1 (Completada)');
  }
  return this.taskService.getByCategoryAndState(req.user.sub, category_id, state);
}
```

Service (Implementación segura para Oracle)

typescript

```
async getByCategoryAndState(
  userId: number,
  category_id: number,
  state: 0 | 1 // Tipo explícito
): Promise<Task[]> {
  return this.repository.find({
    where: {
      user: { id: userId },
      category: category_id !== -1 ? { id: category_id } : IsNull(),
      state: state // Filtro por estado (ahora sí funcionará)
    },
    order: { name: 'DESC' }
  });
}
```

¿Por qué fallaba antes?

1. Ruta incompleta:

Antes tenías `@Get('category/:category_id')`, pero **no definías `:state`** como parte de la ruta, por lo que NestJS no lo reconocía como parámetro.

2. Tipo incorrecto:

Los `@Param()` llegan como **strings** por defecto, pero tu entidad usa **numbers**.
Solución: Usar `ParseIntPipe` para convertirlos.

3. Validación faltante:

Oracle es estricto con los tipos. Asegúrate de que el `state` sea realmente 0 o 1 antes de enviarlo a la consulta.

Ejemplos de llamadas correctas

- **Tareas sin categoría (`category_id = -1`) y estado 0 (Pendientes):**

`GET /api/tasks/category/-1/0`

- **Tareas de categoría 2 y estado 1 (Completadas):**

`GET /api/tasks/category/2/1`

Bonus: Documentación Swagger mejorada

typescript

```
@ApiParam({
  name: 'state',
  type: 'integer',
  enum: [0, 1], // Muestra valores permitidos
  description: '0: Pendiente | 1: Completada'
})
```

<https://i.imgur.com/XYZswagger.png>

Con estos cambios:

- ☐ El filtrado por estado funcionará
- ☐ La API será más clara y robusta
- ☐ Swagger mostrará correctamente los parámetros

¿Necesitas algún otro ajuste? ☐

Entiendo que esto `where: { id }` es equivalente a esto: `where: { id: id }`, verdad?

¡Exactamente! ☐ En JavaScript/TypeScript (y por ende en TypeORM), estas dos sintaxis son **equivalentes** cuando el nombre de la propiedad y la variable coinciden:

Versión corta (shorthand property):

```
typescript
const id = 5;
const where = { id }; // Equivale a { id: id }
```

Versión explícita:

```
typescript
const where = { id: id };
```

En el contexto de TypeORM:

Ambas formas generarán el mismo SQL (`WHERE id = 5`):

```
typescript
// Opción 1 (Shorthand)
await repository.find({ where: { id } });

// Opción 2 (Explícita)
await repository.find({ where: { id: id } });
```

Casos especiales:

1. Si la variable tiene otro nombre, debes ser explícito:

```
typescript
```

- ```
const userId = 10;
await repository.find({ where: { id: userId } }); // No funciona con { userId }
```

- Para relaciones anidadas, la forma explícita es más clara:

```
typescript
2. // Relación con User
3. await repository.find({
4. where: {
5. user: { id: userId } // Mejor que { user: { id } } si no es
 obvio
6. }
7. });
```

## Conclusión:

Usa la forma corta (`{ id }`) cuando:

- La variable se llame igual que la propiedad de la entidad.
- El código sea fácil de leer (evítala si crea confusión).

Usa la forma explícita (`{ id: id }`) cuando:

- Necesites mayor claridad.
- Los nombres no coincidan exactamente.

¡Ambas son correctas y producen el mismo resultado! □ ¿Quieres que profundice en algún otro aspecto?

const whereCondition: any = { user: { id: userId }, category: category\_id !== -1 ? { id: category\_id } : IsNull(), state }; Tones esto estaría bien?

¡Sí, tu código es **perfectamente válido** y funcionará correctamente! □ Veamos por qué:

### Tu implementación (correcta):

typescript

```
const whereCondition: any = {
 user: { id: userId },
 category: category_id !== -1 ? { id: category_id } : IsNull(),
 state // Shorthand property equivalente a state: state
};
```

### ¿Por qué está bien?

1. **Sintaxis abreviada (state):**
  - Es equivalente a `state: state` (mientras la variable se llame igual que la propiedad).
  - TypeORM lo interpretará como `WHERE state = valor_de_state`.
2. **Consistencia con TypeORM:**
  - Para `user` y `category` usas la forma explícita (necesaria por la lógica condicional).
  - Para `state` usas la forma abreviada (limpia y legible).
3. **Oracle-friendly:**
  - `IsNull()` generará el SQL correcto (`IS NULL`) para Oracle.

### Recomendación opcional (para mejor type-safety):

Si quieres evitar `any` y ser más explícito con los tipos:

typescript

```
import { FindOptionsWhere } from 'typeorm';
import { Task } from '../entities/task.entity'; // Ajusta la ruta

const whereCondition: FindOptionsWhere<Task> = {
 user: { id: userId },
 category: category_id !== -1 ? { id: category_id } : IsNull(),
 state
};
```

### Caso de ejemplo:

Si `userId = 1`, `category_id = -1`, y `state = 0`, la consulta generada será:

sql

```
WHERE
 user.id = 1
 AND category.id IS NULL
```

```
AND state = 0
```

## Conclusión:

Tu implementación actual es:

- ☐ **Correcta**
- ☐ **Legible**
- ☐ **Funcional para Oracle**

¿Necesitas ajustar algo más? ¡Estoy aquí para ayudar! ☐

**Cuando hay tarea me retorna vacío, qué está pasando? `async getById(id: number): Promise<Task | null> { return this.repository.findOne({ where: { id }, relations: ['user'] }) }`**

El problema de que `getById` retorne vacío (`null`) a pesar de que exista la tarea puede deberse a varias causas. Vamos a analizarlo paso a paso:

## Posibles causas y soluciones:

### 1. El ID no existe en la base de datos

typescript

```
• // Verifica manualmente en tu servicio:
async getById(id: number) {
 const exists = await this.repository.exist({ where: { id } });
 if (!exists) throw new NotFoundException('Task not found');
 return this.repository.findOne({ where: { id }, relations: ['user']
});
}
```

- **Problema con la relación `user`** (si es opcional y no está definida)

typescript

```
• // Opción 1: Hacer la relación opcional
return this.repository.findOne({
 where: { id },
 relations: ['user'],
 loadEagerRelations: false // Ignora si la relación no existe
});

// Opción 2: Usar leftJoin
return this.repository.createQueryBuilder('task')
 .leftJoinAndSelect('task.user', 'user')
 .where('task.id = :id', { id })
 .getOne();
```

- **Problema de tipos en el ID** (especialmente con Oracle)

typescript

```
• // Asegúrate de convertir el ID a número
async getById(@Param('id', ParseIntPipe) id: number) {
```

```

return this.repository.findOne({
 where: { id: () => `TO_NUMBER(${id})` }, // Para Oracle
 relations: ['user']
});
}

```

- **Cache o transacciones no comprometidas**

typescript

```

4. // Prueba con una consulta RAW
5. async getById(id: number) {
6. const [task] = await this.repository.query(
7. 'SELECT * FROM tasks WHERE id = ?',
8. [id]
9.);
10. return task || null;
11. }

```

## Diagnóstico rápido:

1. **Verifica datos directamente en la BD:**

sql

```

• SELECT * FROM tasks WHERE id = [tu_id];
SELECT * FROM users WHERE id = [user_id_de_la_tarea];

```

- **Prueba sin relaciones:**

typescript

```

• // Test básico sin joins
async getById(id: number) {
 return this.repository.findOneBy({ id }); // Sin relations
}

```

- **Habilita logging de TypeORM (en tu módulo):**

typescript

```

3. TypeOrmModule.forRoot({
4. logging: true, // Verás la consulta SQL generada
5. })

```

## Solución completa (adaptada a Oracle):

typescript

```

async getById(id: number): Promise<Task | null> {
 return this.repository.createQueryBuilder('task')
 .leftJoinAndSelect('task.user', 'user')
 .where('task.id = :id', { id })
 // Para Oracle NUMBER:
 // .where('task.id = TO_NUMBER(:id)', { id })
 .getOne();
}

```

Si el problema persiste, comparte:

- La estructura de tu entidad `Task` (especialmente las columnas `id` y `user`)
- El SQL que muestra el logging de TypeORM
- La versión exacta de tu base de datos Oracle

**async getById(id: number): Promise<Task | null> { return this.repository.findOne({ where: { id }, relations: ['user', 'category'], loadEagerRelations: false }) } por qué solo me devuelve el id y nombre de la categoría y no el resto de cosas?**

El problema de que solo obtengas el `id` y `name` de la categoría (y no el resto de los campos) se debe a cómo TypeORM maneja las relaciones y la serialización. Aquí está la solución detallada:

### Causa del problema:

1. **Serialización parcial:**
  - TypeORM puede estar aplicando algún decorador (`@Exclude()`, `class-transformer`) que limita los campos devueltos.
2. **Configuración de la relación:**
  - La relación `category` podría estar definida sin seleccionar todos los campos.

### Soluciones:

#### 1. Verifica tu entidad `Category`:

Asegúrate de que todos los campos estén definidos y no tengan decoradores que los excluyan:

```
typescript
@Entity()
export class Category {
 @PrimaryGeneratedColumn()
 id: number;

 @Column()
 name: string;

 @Column() // Asegúrate de que todos los campos estén decorados
 description: string;

 @Column()
 color: string;

 // ... otros campos
}
```

#### 2. Usa `QueryBuilder` para control explícito:

```
typescript
async getById(id: number): Promise<Task | null> {
 return this.repository.createQueryBuilder('task')
 .leftJoinAndSelect('task.user', 'user')
}
```



```

 .leftJoinAndSelect('task.category', 'category')
 .select([
 'task',
 'user.id',
 'user.name',
 'category.id',
 'category.name',
 'category.description',
 'category.color' // Añade todos los campos necesarios
])
 .where('task.id = :id', { id })
 .findOne();
 }
}

```

### 3. Configuración global de TypeORM (opcional):

En tu módulo principal:

typescript

```

TypeOrmModule.forRoot({
 // ...
 logging: true, // Para ver la consulta SQL generada
 synchronize: false,
 extra: {
 // Para Oracle (si es tu BD)
 fetchAsString: ['clob', 'number']
 }
})

```

### 4. Si usas class-transformer:

Verifica que no tengas decoradores que excluyan campos:

typescript

```

export class Category {
 // ❌ Esto excluiría el campo:
 // @Exclude()
 @Column()
 description: string;
}

```

### Diagnóstico adicional:

1. **Revisa el SQL generado** (habilita `logging: true`).
2. **Prueba una consulta RAW** para verificar los datos:

typescript

```

const data = await this.repository.query(
 'SELECT * FROM categories WHERE id = ?',
 [categoryId]
);
console.log(data);

```

### Solución definitiva (probable causa):

Lo más común es que en tu entidad `Task`, la relación con `Category` esté configurada sin `eager: true` y con carga parcial. Cambia esto:

```
typescript
@Entity()
export class Task {
 // ...

 @ManyToOne(() => Category, { eager: true }) // ← Añade esto
 @JoinColumn({ name: 'category_id' })
 category: Category;
}
```

Si el problema persiste, comparte:

1. La estructura completa de tu entidad `Category`.
2. El SQL que muestra el logging (con `logging: true`).
3. La versión exacta de TypeORM que usas.

¡Con esto deberías obtener todos los campos de la categoría! □

**El taskData sale undefined, me huele que no se está asignando bien, por qué?**

```
updateState(userId: number, id: number) { const Promise =
this.validateTask(userId, id); let taskData Promise.then((result) => { result.state
== 0? result.state = 1: result.state = 0; taskData = result console.log("el result")
console.log(result) }) console.log("el taskdata") console.log(taskData) return
this.taskRepository.updateState(id, taskData) } el taskdata undefined [Nest] 13560
- 20/07/2025, 22:04:06 ERROR [ExceptionsHandler] UpdateValuesMissingError:
Cannot perform update query because update values are not defined. Call
"qb.set(...)" method to specify updated values. at
UpdateQueryBuilder.getValueSet (C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\node_modules\typeorm\query-builder\src\query-
builder\UpdateQueryBuilder.ts:774:15) at
UpdateQueryBuilder.createUpdateExpression
(C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\node_modules\typeorm\query-builder\src\query-
builder\UpdateQueryBuilder.ts:476:32) at UpdateQueryBuilder.getQuery
(C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\node_modules\typeorm\query-builder\src\query-
builder\UpdateQueryBuilder.ts:53:21) at
UpdateQueryBuilder.getQueryAndParameters
(C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\node_modules\typeorm\query-builder\src\query-
builder\QueryBuilder.ts:495:28) at UpdateQueryBuilder.execute
(C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\node_modules\typeorm\query-builder\src\query-
builder\UpdateQueryBuilder.ts:142:50) at process.processTicksAndRejections
(node:internal/process/task_queues:95:5) at async TaskRepository.updateState
(C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\src\task\repositories\task.repository.ts:53:9) at async
C:\Users\Usuario\Desktop\Proyecto
```

```
A3SATEL\Backend\3TT\node_modules\@nestjs\core\router\router-execution-context.js:46:28 at async C:\Users\Usuario\Desktop\Proyecto
A3SATEL\Backend\3TT\node_modules\@nestjs\core\router\router-proxy.js:9:17
el result Task { id: 5, name: 'Banana', description: 'BANANAAAAAAAAAAAA',
state: 1, user: Users { id: 81, email: 'rafa@ejemplo.com', name: 'rafa', password:
'$2b$10$Q18E0ZYxG0msGdc.UGaRZePYTyt.llqZ.vUzIjeMKR7SFvYxffm9q' },
category: Category { id: 42, name: 'Tareas personales', user: undefined } }
```

El problema es un **clásico error de asincronía** en JavaScript. Aquí está lo que ocurre:

### ❑ Problema Detectado

1. **taskData es undefined** porque estás intentando usarla **antes** de que se resuelva la Promise.
2. El `console.log("el taskdata")` se ejecuta **antes** que el código dentro del `.then()`.
3. Por eso al llamar a `updateState()`, `taskData` aún no está definido.

### ❑ Solución Corregida (usando async/await)

typescript

```
async updateState(userId: number, id: number) {
 try {
 const result = await this.validateTask(userId, id); // Espera la
promesa

 result.state = result.state === 0 ? 1 : 0; // Cambia el estado
 console.log("Datos actualizados:", result);

 return this.taskRepository.updateState(id, result); // Pasa el
objeto completo
 } catch (error) {
 console.error("Error en updateState:", error);
 throw error; // Re-lanza el error para que Nest lo maneje
 }
}
```

### ❑ Cambios Clave:

1. Usa **async/await** en lugar de `.then()` para código más lineal.
2. **Elimina la variable temporal** `taskData` (innecesaria).
3. **Pasa result directamente** al repositorio.

### ❑ Ajusta también tu Repository:

typescript

```
// task.repository.ts
async updateState(id: number, taskData: Partial<Task>) {
 await this.createQueryBuilder()
 .update(Task)
 .set({ state: taskData.state }) // Asegúrate de usar el campo
correcto
 .where("id = :id", { id })
 .execute();
}
```

```
return this.findOneBy({ id }); // Devuelve la tarea actualizada
}
```

### ❑ ¿Por qué pasaba?

- **JavaScript es asíncrono:** El código dentro de `.then()` se ejecuta después del flujo principal.
- **Orden de ejecución:**

javascript

- 1. `console.log("el taskdata")` // ← Primero (taskData es undefined)
- 2. Luego se ejecuta el `.then()`


### ❑ Mejora Adicional (DTO para claridad):

typescript

```
// update-state.dto.ts
export class UpdateStateDto {
 @IsIn([0, 1]) // Valida que sea 0 o 1
 state: number;
}

// En el controller
@Patch('/:id/state')
updateState(
 @Param('id') id: number,
 @Body() dto: UpdateStateDto // Usa el DTO
) {
 return this.taskService.updateState(id, dto.state);
}
```

Con estos cambios, tu método `updateState` funcionará correctamente. ¡El error ya no aparecerá! ❑

**async updateCategory(id: number, taskData: UpdateTaskCategoryDto) { await this.repository.update(id, taskData); return this.repository.findOne({ where: { id }, relations: ['user', 'category'], loadEagerRelations: false }); } Type 'UpdateTaskCategoryDto' has no properties in common with type '\_QueryDeepPartialEntity<Task>'.ts(2559)  [Generate docs \(Ctrl+.\)](#) (parameter) taskData: UpdateTaskCategoryDto**

El error ocurre porque `UpdateTaskCategoryDto` no coincide con la estructura esperada por `repository.update()`. Vamos a solucionarlo:

### Problema:

TypeORM espera un objeto que coincida con parciales de tu entidad `Task` (`_QueryDeepPartialEntity<Task>`), pero tu DTO no está correctamente alineado.

### Solución:

## 1. Asegura que el DTO refleje campos de la entidad:

typescript

```
// update-task-category.dto.ts
export class UpdateTaskCategoryDto {
 @IsNumber()
 @IsOptional()
 categoryId?: number; // Asegúrate que coincida con el nombre en Task
}
```

## 2. Actualiza el servicio para mapear el DTO:

typescript

```
async updateCategory(id: number, taskData: UpdateTaskCategoryDto) {
 // Mapea el DTO a la estructura de Task
 const updateData = {
 category: taskData.categoryId ? { id: taskData.categoryId } : null
 };

 await this.repository.update(id, updateData);

 return this.repository.findOne({
 where: { id },
 relations: ['user', 'category'],
 loadEagerRelations: false
 });
}
```

## Alternativa mejor (usando QueryBuilder):

typescript

```
async updateCategory(id: number, taskData: UpdateTaskCategoryDto) {
 await this.repository
 .createQueryBuilder()
 .update(Task)
 .set({
 category: taskData.categoryId ? { id: taskData.categoryId } :
null
 })
 .where("id = :id", { id })
 .execute();

 return this.repository.findOne({
 where: { id },
 relations: ['user', 'category']
 });
}
```

## ¿Por qué funciona?

- **TypeORM necesita el formato exacto de tu entidad:**
  - Si Task tiene @ManyToOne(() => Category), debes pasar { category: { id: X } }
  - No basta con { categoryId: X }

## Recomendación adicional:

```
typescript
// Entidad Task.ts
@Entity()
export class Task {
 // ...

 @ManyToOne(() => Category, { nullable: true })
 @JoinColumn({ name: 'category_id' }) // Asegúrate que coincida con
 tu DB
 category: Category | null;

 // Opcional: Si quieres acceder directo al ID
 @Column({ nullable: true })
 category_id?: number;
}
```

Si sigues teniendo problemas, comparte:

1. La estructura exacta de tu `UpdateTaskCategoryDto`
2. La definición de la entidad `Task` (especialmente la parte de `category`)
3. La versión de TypeORM que usas

**como gestiono para que cambie la categoría a null si el número recibido es -1?**

```
async updateCategory(userId: number, id: number, updateTaskCategoryDto:
UpdateTaskCategoryDto) { const result = await this.validateTask(userId, id); const
taskData = {category: { id: updateTaskCategoryDto.categoryId }} return
this.taskRepository.updateCategory(id, taskData) } async updateCategory(id:
number, taskData: {category: {id: number}}) { await this.repository.update(id,
taskData); return this.repository.findOne({ where: { id }, relations: ['user',
'category'], loadEagerRelations: false }); }
```

Para gestionar correctamente el cambio de categoría a `null` cuando recibes `-1`, aquí tienes la solución completa:

## 1. Primero modifica tu DTO para permitir explícitamente el valor -1:

```
typescript
// update-task-category.dto.ts
export class UpdateTaskCategoryDto {
 @IsNumber()
 @Min(-1) // Permite -1 como valor válido
 categoryId: number;
}
```

## 2. Luego actualiza tu servicio para manejar el -1:

```
typescript
async updateCategory(userId: number, id: number,
updateTaskCategoryDto: UpdateTaskCategoryDto) {
 await this.validateTask(userId, id);

 const updateData = {
 category: updateTaskCategoryDto.categoryId === -1
 ? null
 : { id: updateTaskCategoryDto.categoryId }
 }
```

```
};

return this.taskRepository.updateCategory(id, updateData);
}
```

### 3. Finalmente ajusta el repositorio:

typescript

```
async updateCategory(id: number, taskData: { category: { id: number }
| null }) {
 // Usa QueryBuilder para mejor control con Oracle
 await this.repository
 .createQueryBuilder()
 .update(Task)
 .set({
 category: taskData.category
 })
 .where("id = :id", { id })
 .execute();

 return this.repository.findOne({
 where: { id },
 relations: ['user', 'category'],
 loadEagerRelations: false
 });
}
```

### ¿Por qué esta solución funciona mejor?

1. **Manejo explícito de -1:** Convierte claramente -1 a null
2. **Type-safe:** El tipo { category: { id: number } | null } refleja exactamente lo que esperas
3. **Oracle compatible:** El QueryBuilder genera el SQL adecuado para NULLs

### Ejemplo de uso:

- **Para asignar categoría 5:**

json

- { "categoryId": 5 }

Genera: category = { id: 5 }

- **Para quitar categoría (null):**

json

- { "categoryId": -1 }

Genera: category = null

### Bonus: Validación adicional en el servicio

typescript

```
if (updateTaskCategoryDto.categoryId !== -1) {
 await this.validateCategoryExists(updateTaskCategoryDto.categoryId);
}
```

Esta implementación manejará correctamente:

- ☐ Asignación de categoría existente
- ☐ Eliminación de categoría (con -1)
- ☐ Relaciones en TypeORM
- ☐ Compatibilidad con Oracle