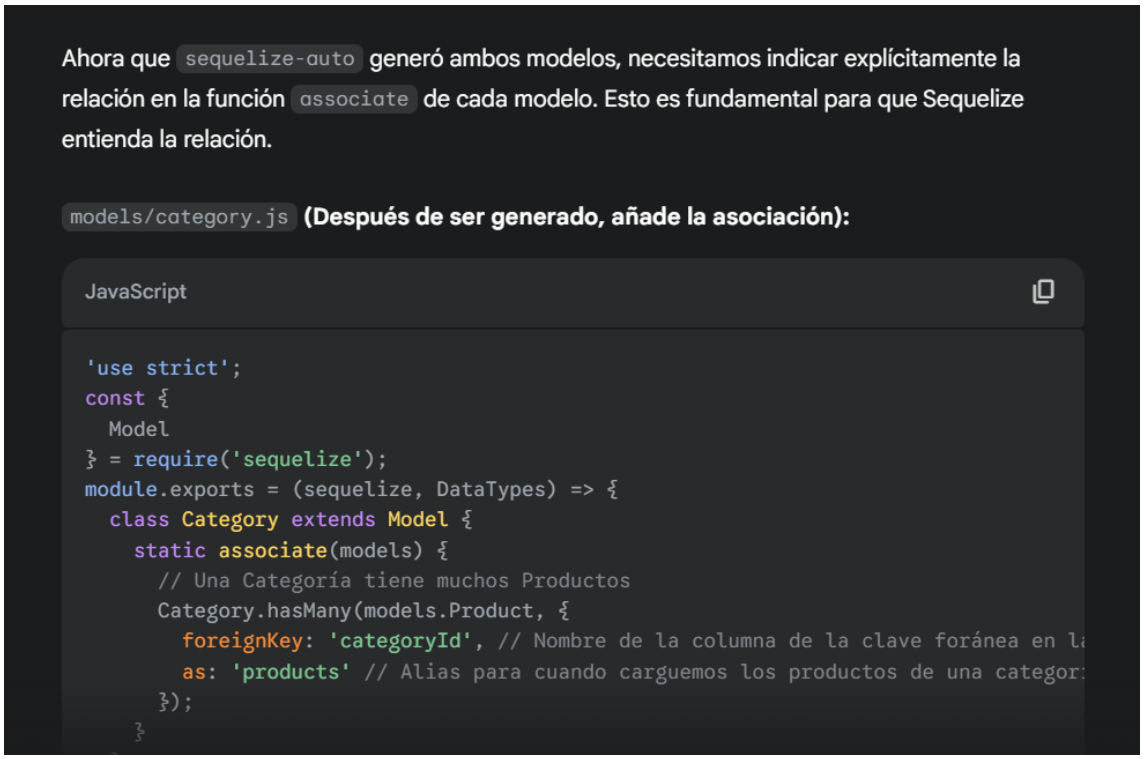
```
-- Crear la tabla Categories
CREATE TABLE IF NOT EXISTS "Categories" (
  "id" SERIAL PRIMARY KEY,
  "name" VARCHAR(255) NOT NULL UNIQUE,
  "description" TEXT,
  "createdAt" TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
  "updatedAt" TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Insertar algunas categorías de prueba
INSERT INTO "Categories" (name, description) VALUES
('Electrónica', 'Dispositivos y gadgets electrónicos.'),
('Hogar y Cocina', 'Artículos para el hogar y la cocina.'),
('Libros', 'Libros de diferentes géneros.'),
('Ropa', 'Artículos de vestir para hombres y mujeres.');

-- Añadir la columna categoryId a la tabla Products
ALTER TABLE "Products"
ADD COLUMN "categoryId" INTEGER;

-- Añadir la restricción de clave foránea
ALTER TABLE "Products"
ADD CONSTRAINT "fk_category"
FOREIGN KEY ("categoryId")
REFERENCES "Categories"("id")
ON DELETE SET NULL; -- Opciones: CASCADE, RESTRICT, NO ACTION, SET DEFAULT, SET NULL

-- Actualizar productos existentes para que tengan una categoría (opcional)
UPDATE "Products" SET "categoryId" = (SELECT id FROM "Categories" WHERE name = 'Electrónica')
WHERE name IN ('Laptop Pro', 'Smart TV 4K', 'Auriculares Bluetooth');
UPDATE "Products" SET "categoryId" = (SELECT id FROM "Categories" WHERE name = 'Hogar y
Cocina') WHERE name LIKE '%Licuadora%'; -- Si tienes algún producto de cocina

-- Asegurarse de que categoryId pueda ser NULL para productos existentes que no se actualicen
-- ALTER TABLE "Products" ALTER COLUMN "categoryId" DROP NOT NULL;
```




```
'use strict';

const {
 Model
} = require('sequelize');
module.exports = (sequelize, DataTypes) => {
 class Category extends Model {
  static associate(models) {
   // Una Categoría tiene muchos Productos
   Category.hasMany(models.Product, {
    foreignKey: 'categoryId', // Nombre de la columna de la clave foránea en la tabla Products
    as: 'products' // Alias para cuando carguemos los productos de una categoría
   });
  }
 }
 Category.init({
  id: { // sequelize-auto puede generarlo con este tipo si es SERIAL
   type: DataTypes.INTEGER,
   allowNull: false,
   primaryKey: true,
   autoIncrement: true
  },
  name: {
   type: DataTypes.STRING,
   allowNull: false,
   unique: true // Asegúrate de que coincida con la DB
  },
  description: {
   type: DataTypes.TEXT,
   allowNull: true
  },
  createdAt: {
   type: DataTypes.DATE,
```

```
    allowNull: false,
    field: 'createdAt' // Asegúrate de que el nombre del campo coincida con la DB si es camelCase
   },
   updatedAt: {
    type: DataTypes.DATE,
    allowNull: false,
    field: 'updatedAt'
   }
  }, {
   sequelize,
   modelName: 'Category',
   tableName: 'Categories', // Asegúrate de que coincida con tu tabla
   timestamps: true, // Si tu tabla tiene createdAt y updatedAt
   underscored: false // Si tus columnas son camelCase (createdAt, updatedAt)
  });
  return Category;
};
```


```
'use strict';
const {
 Model
} = require('sequelize');
module.exports = (sequelize, DataTypes) => {
 class Product extends Model {
  static associate(models) {
   // Un Producto pertenece a una Categoría
   Product.belongsTo(models.Category, {
    foreignKey: 'categoryId', // Nombre de la columna de la clave foránea en la tabla Products
    as: 'category' // Alias para cuando carguemos la categoría de un producto
   });
  }
 }
 Product.init({
  id: { // sequelize-auto puede generarlo con este tipo si es SERIAL
   type: DataTypes.INTEGER,
   allowNull: false,
```

```
    primaryKey: true,
    autoIncrement: true
   },
   name: {
    type: DataTypes.STRING,
    allowNull: false
   },
   description: {
    type: DataTypes.TEXT,
    allowNull: true
   },
   price: {
    type: DataTypes.DECIMAL(10, 2),
    allowNull: false
   },
   categoryId: { // Asegúrate de que esta columna esté definida y sea INTEGER
    type: DataTypes.INTEGER,
    allowNull: true // Puede ser null si la clave foránea lo permite (ON DELETE SET NULL)
   },
   createdAt: {
    type: DataTypes.DATE,
    allowNull: false,
    field: 'createdAt'
   },
   updatedAt: {
    type: DataTypes.DATE,
    allowNull: false,
    field: 'updatedAt'
   }
  }, {
   sequelize,
   modelName: 'Product',
   tableName: 'Products', // Asegúrate de que coincida con tu tabla
   timestamps: true,
   underscored: false // Si tus columnas son camelCase (createdAt, updatedAt)
  });
  return Product;
};
```

4. Actualizar Controladores para Usar Asociaciones

Ahora podemos usar las asociaciones en nuestras consultas para cargar datos relacionados.

`controllers/productController.js` **(Ejemplos con asociación):**

```
const db = require('../models');
const Product = db.Product;
const Category = db.Category; // Importar el modelo Category

// Crear un nuevo producto
exports.create = async (req, res) => {
 try {
  const { name, description, price, categoryId } = req.body;
  if (!name || !price) {
   return res.status(400).send({ message: 'El nombre y el precio son campos requeridos.' });
  }

  // Opcional: Validar que la categoría exista
  if (categoryId) {
   const category = await Category.findByPk(categoryId);
   if (!category) {
    return res.status(404).send({ message: `La categoría con ID ${categoryId} no existe.` });
   }
  }

  const product = await Product.create({ name, description, price, categoryId });
  res.status(201).send(product);
 } catch (error) {
  console.error('Error al crear el producto:', error);
  res.status(500).send({
   message: error.message || 'Ocurrió un error al crear el producto.'
  });
 }
};

// Obtener todos los productos (incluyendo la categoría)
exports.findAll = async (req, res) => {
 try {
  const products = await Product.findAll({
   include: {
    model: Category,
    as: 'category', // Usa el alias definido en la asociación (belongsTo)
    attributes: ['id', 'name'] // Solo incluye id y nombre de la categoría
   }
  });
  res.status(200).send(products);
 } catch (error) {
  console.error('Error al recuperar los productos:', error);
  res.status(500).send({
   message: error.message || 'Ocurrió un error al recuperar los productos.'
  });
 }
};

// Obtener un producto por ID (incluyendo la categoría)
exports.findOne = async (req, res) => {
 const id = req.params.id;
 try {
  const product = await Product.findByPk(id, {
   include: {
    model: Category,
    as: 'category', // Usa el alias definido
```

```
      attributes: ['id', 'name', 'description']
    }
  });
  if (!product) {
    return res.status(404).send({ message: `No se encontró un producto con id=${id}.` });
  }
  res.status(200).send(product);
 } catch (error) {
  console.error('Error al recuperar el producto por ID:', error);
  res.status(500).send({
    message: 'Error al recuperar el producto con id=' + id
  });
 }
};

// ... (Las funciones update, delete, deleteAll permanecen similares o puedes expandirlas para
manejar categoryId)
// Por ejemplo, para update, puedes permitir actualizar categoryId
exports.update = async (req, res) => {
  const id = req.params.id;
  const { name, description, price, categoryId } = req.body;

  try {
    if (categoryId) {
      const category = await Category.findByPk(categoryId);
      if (!category) {
        return res.status(404).send({ message: `La categoría con ID ${categoryId} no existe.` });
      }
    }

    const [num] = await Product.update({ name, description, price, categoryId }, {
      where: { id: id }
    });

    if (num === 1) {
      res.status(200).send({ message: 'El producto se actualizó correctamente.' });
    } else {
      res.status(404).send({
        message: `No se puede actualizar el producto con id=${id}. Tal vez el producto no fue
encontrado o req.body está vacío.`
      });
    }
  } catch (error) {
    console.error('Error al actualizar el producto:', error);
    res.status(500).send({
      message: 'Error al actualizar el producto con id=' + id
    });
  }
};
```

Crear un nuevo controlador para Categorías (`controllers/categoryController.js`):

```
const db = require('../models');
const Category = db.Category;
const Product = db.Product; // También necesitamos el modelo Producto

// Crear una nueva categoría
exports.create = async (req, res) => {
 try {
  const { name, description } = req.body;
  if (!name) {
   return res.status(400).send({ message: 'El nombre de la categoría es requerido.' });
  }
  const category = await Category.create({ name, description });
  res.status(201).send(category);
 } catch (error) {
  console.error('Error al crear la categoría:', error);
  res.status(500).send({
   message: error.message || 'Ocurrió un error al crear la categoría.'
  });
 }
};

// Obtener todas las categorías (opcionalmente con sus productos)
exports.findAll = async (req, res) => {
 const includeProducts = req.query.includeProducts === 'true'; // Parámetro de query para incluir
productos
 try {
  const options = {};
  if (includeProducts) {
   options.include = {
    model: Product,
    as: 'products', // Usa el alias definido en la asociación (hasMany)
    attributes: ['id', 'name', 'price']
   };
  }
  const categories = await Category.findAll(options);
  res.status(200).send(categories);
 } catch (error) {
  console.error('Error al recuperar las categorías:', error);
  res.status(500).send({
   message: error.message || 'Ocurrió un error al recuperar las categorías.'
  });
 }
};

// Obtener una categoría por ID (opcionalmente con sus productos)
exports.findOne = async (req, res) => {
 const id = req.params.id;
 const includeProducts = req.query.includeProducts === 'true';
 try {
  const options = {};
  if (includeProducts) {
   options.include = {
```

```
      model: Product,
      as: 'products',
      attributes: ['id', 'name', 'price']
     };
   }
   const category = await Category.findByPk(id, options);
   if (!category) {
    return res.status(404).send({ message: `No se encontró una categoría con id=${id}.` });
   }
   res.status(200).send(category);
  } catch (error) {
   console.error('Error al recuperar la categoría por ID:', error);
   res.status(500).send({
    message: 'Error al recuperar la categoría con id=' + id
   });
  }
};

// Actualizar una categoría
exports.update = async (req, res) => {
 const id = req.params.id;
 try {
  const [num] = await Category.update(req.body, {
   where: { id: id }
  });

  if (num === 1) {
   res.status(200).send({ message: 'La categoría se actualizó correctamente.' });
  } else {
   res.status(404).send({
    message: `No se puede actualizar la categoría con id=${id}. Tal vez la categoría no fue
encontrada o req.body está vacío.`
   });
  }
 } catch (error) {
  console.error('Error al actualizar la categoría:', error);
  res.status(500).send({
   message: 'Error al actualizar la categoría con id=' + id
  });
 }
};

// Eliminar una categoría
exports.delete = async (req, res) => {
 const id = req.params.id;
 try {
  const num = await Category.destroy({
   where: { id: id }
  });

  if (num === 1) {
   res.status(200).send({ message: 'La categoría se eliminó correctamente!' });
  } else {
   res.status(404).send({
    message: `No se pudo eliminar la categoría con id=${id}. Tal vez la categoría no fue
encontrada!`
   });
```

```
    }
  } catch (error) {
    console.error('Error al eliminar la categoría:', error);
    res.status(500).send({
      message: 'No se pudo eliminar la categoría con id=' + id
    });
  }
};
```

5. Definir las Rutas para Categorías

Crea un nuevo archivo `routes/categoryRoutes.js`:

`routes/categoryRoutes.js`:

```
const express = require('express');
const router = express.Router();
const categories = require('../controllers/categoryController.js');

// Rutas para las operaciones CRUD en categorías

// Crear una nueva categoría
router.post('/', categories.create);

// Obtener todas las categorías (con o sin productos relacionados)
router.get('/', categories.findAll);

// Obtener una categoría por ID (con o sin productos relacionados)
router.get('/:id', categories.findOne);

// Actualizar una categoría por ID
router.put('/:id', categories.update);

// Eliminar una categoría por ID
router.delete('/:id', categories.delete);

module.exports = router;
```

6. Integrar las Nuevas Rutas en `app.js`

Finalmente, añade las rutas de categoría a tu aplicación principal.

`app.js` **(Fragmento):**

```
// Rutas de productos (existente)
const productRoutes = require('./routes/productRoutes');
app.use('/api/products', productRoutes);

// Nuevas rutas de categorías
const categoryRoutes = require('./routes/categoryRoutes');
app.use('/api/categories', categoryRoutes);
```

- **URL:** `http://localhost:3000/api/categories`
- **Method:** `POST`
- **Body (raw, JSON):**

JSON

```
{
    "name": "Deportes",
    "description": "Artículos para actividades deportivas."
}
```

2. Crear un Producto con una Categoría (POST)

- **URL:** `http://localhost:3000/api/products`
- **Method:** `POST`
- **Body (raw, JSON):**

JSON

```
{
    "name": "Balón de Fútbol",
    "description": "Balón oficial para partidos y entrenamiento.",
    "price": 35.00,
    "categoryId": 5 // Usa el ID de una categoría existente (ej. la de "Deporte
}
```