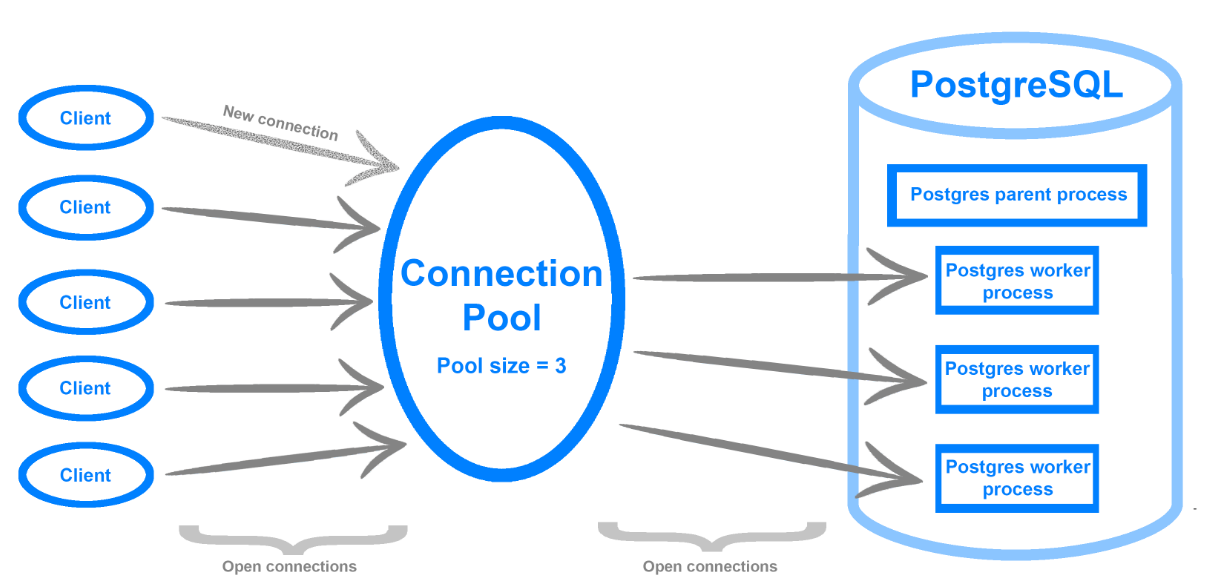
# **Pool de conexiones**

Un pool de conexiones **es un conjunto limitado de conexiones a una base de datos**, que es manejado por un servidor de aplicaciones de forma tal, que dichas conexiones pueden ser reutilizadas por los diferentes usuarios.



# **Variables de ambiente en Node.js**

Las variables de entorno se utilizan para contener contenido sensible

## **Instalamos el siguiente paquete**

Esto nos sirve para tener las variables de entorno corriendo en el proceso de node.



Es una buena práctica tener un archivo **.env** y un **.env.example** (aquí se colocaran variables de entorno de ejemplo para otros programadores)

# **¿Qué es un ORM?**

Un ORM (Object Relational Model) es un modelo de programación que permite mapear las estructuras de una base de datos relacionales.

Al abstraer este tipo de programación, delegamos su implementación al backend, es decir, le añadimos una de responsabilidad a la capa transaccional del servidor.

Transforma y mapea nuestra BD con métodos de la programación orientada a objetos. Se pueden ejecutar métodos en lugar de consultas.

Hay 2 ORM muy populares, se recomienda [Sequelize](https://sequelize.org/) para JavaScript y [TypeORM](https://typeorm.io/) para TypeScript.

## **Ventajas**

* Es **agnóstico**, es decir, no importa si se está usando MySQL, PostgreSQL, MariaDB, no hay problema mientras la BD use SQL.
* Acciones como CRUD (Create, Read, Update, Delete) son administradas mediante ORM.
* La implementación de **seeds** o semillas, nos permiten recuperar, mediante código, la estructura de una BD.

Una de las bases teóricas para entender este modelo es mediante el conocimiento de **DAO** (Data Access Object) y **DTO** (Data Transfer Object), los cuales nos permiten desestructurar un ORM en módulos de abstracción para acceder a la DB y transferir datos desde la misma DB, respectivamente hablando.

## **Desventajas**

* Delegación de responsabilidades al server.
* Descentralización de trabajo, directa, de una BD.

# **¿Qué son las migraciones? Migraciones en Sequelize ORM**

Distintas definiciones en:

## **Django**

Las migraciones son la forma en que Django **propaga cambios** en los modelos y los refleja en el esquema de bases de datos.

## **Laravel**

Las migraciones son como un sistema de **control de versiones** para la base de datos.

## **Sequelize**

Es como un sistema de **control de versiones** para manejar los cambios desde el código y trackear los cambios en la base de datos.

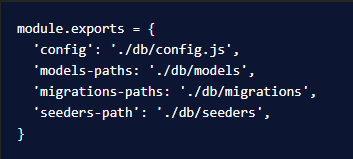
Básicamente, las migraciones mantienen el historial del esquema que se lleva en la base de datos. Es un sistema muy usado en ambientes de producción para **trackear los cambios** sin tener que replicar todo nuevamente (creación de tablas, llaves foráneas, etc). Es decir, permite saber en **qué punto** **estaba** para saber qué es lo que se tiene que modificar.

**sequelize.sync()** empieza a leer los modelos, crea tablas y hace relist (se sobrescribe información), **no se aconseja que se corra en producción**. Es mejor sincronizar con un sistema de migraciones.

Para correr migraciones se utiliza la librería **sequelize-cli** y se instala como dependencia de desarrollo con el comando **npm i sequelize-cli -D**.

Posteriormente, se crea un archivo de configuración **.sequelizerc** en la carpeta principal.

## **.sequelizerc**



* **config**: Dónde se encuentra la configuración, esta configuración se encuentra la conexión hacia la BD. El **cli** tiene su propia conexión, independientemente de la conexión de la aplicación porque esas conexiones corren a nivel de terminal.
* **models-paths**: Dónde se encuentran los modelos.
* **migrations-paths**: Dónde se encuentran las migraciones.
* **seeders-path**: Dónde se encuentran las semillas de información, sirve mucho para pruebas unitarias, **end to end**, donde se necesitan semillas de información que es como cargar varios datos de información a la BD.

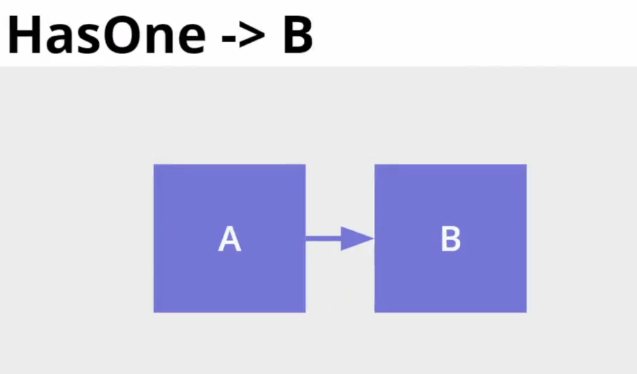
Se crean las carpetas **migrations**, **models**, **seeders** y el archivo **config.js** dentro de la carpeta **db**.

# **Relaciones uno a uno**

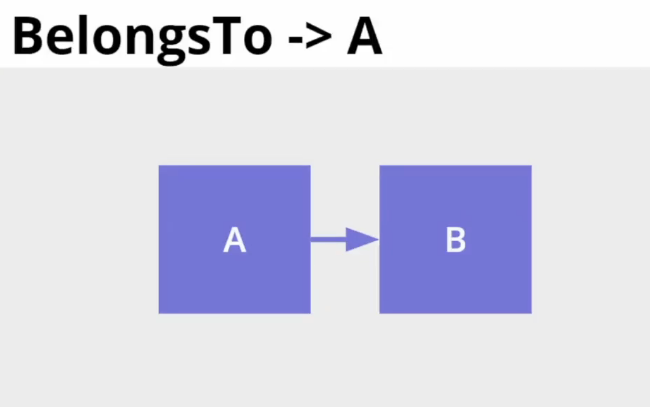
Sequelize tiene dos métodos para crear la relación uno a uno.

* **hasOne**.
* **belongsTo**.

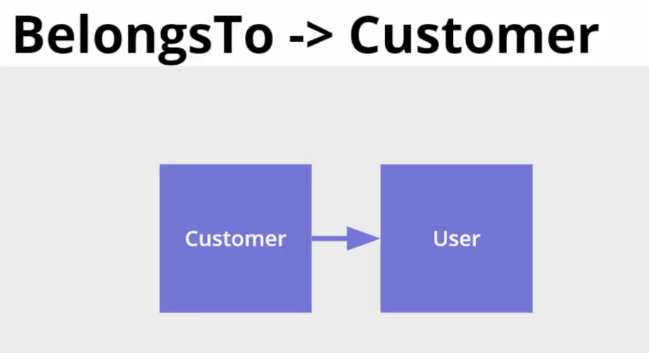
Sí queremos que la relación quede en la entidad **B**, deberíamos utilizar **hasOne**.



Por lo contrario, sí queremos que la relación quede en la entidad **A**, utilizamos **belongsTo**.



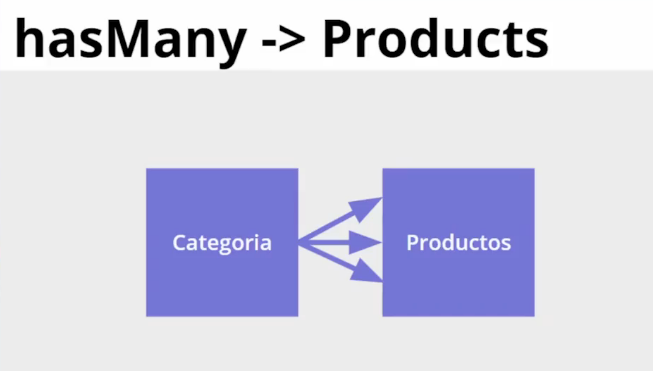
En las entidades de **Costumer** y **User**, sí queremos que la relación sea cargada desde **Costumer** entonces utilizaríamos un **belongsTo**.



# **Relaciones uno a muchos**

Sequelize tiene un método para crear la relación uno a muchos.

* **hasMany**



# **Relaciones muchos a muchos**

Sequelize tiene un método para crear la relaciones muchos a muchos.

* **belongsToMany**

