**Más que una herramienta**

*“A veces, las cosas no pueden ir a tu manera, pero el esfuerzo debe estar ahí cada noche” Michael Jordan — Ex jugador de baloncesto*

**Introducción**

Como viste en las meetings anteriores, las b[ases de datos](https://es.wikipedia.org/wiki/Base_de_datos) tienen su propio [lenguaje de programación](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n) para realizar consultas y así poder guardar y obtener sus datos, por ejemplo: todas las base de datos relacionales usan SQL como lenguaje de programación base, y cada sistema le agrega su toque distintivo, como pueden ser: [MySQL](https://es.wikipedia.org/wiki/MySQL), [SQL Server](https://es.wikipedia.org/wiki/Microsoft_SQL_Server) o [PostgreSQL](https://es.wikipedia.org/wiki/PostgreSQL).



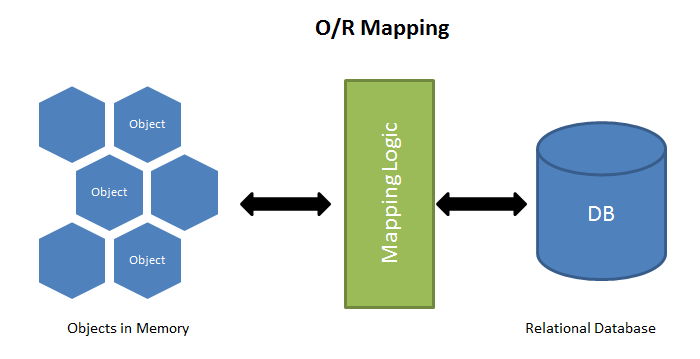
Cuando estás programando una aplicación es muy común que necesites hacer uso de alguna base de datos para guardar información, si este es el caso puede que te preguntes: ¿qué base de datos me conviene usar?, ¿cuál es mejor? o ¿por qué necesito saber SQL si yo solamente quiero guardar datos en mi aplicación?

Antes que nada, si tienes dudas, pregunta, siempre hay alguien dispuesto a ayudarte a resolver tus problemas. Como ya sabes, los/as developers trabajamos muy estrechamente en comunidad con otros/as desarrolladores. Habiendo dicho esto, no hay una base de datos que sea mejor o peor, son las características de cada una la que las diferencian. Con esto en mente, podemos decidir cuál es más apropiada según lo que necesites hacer.

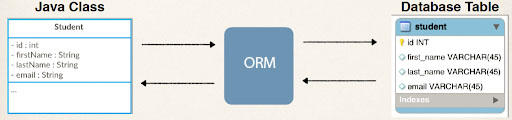
¡Ahora viene lo mejor! En esta toolbox vamos ver una gran técnica que te va a ser muy útil a la hora de programar, si es que necesitas hacer uso de algunas base de datos. Démosle la bienvenida a ORM.

**Primero lo primero: ¿qué es un ORM?**

Ya tuviste un primer acercamiento un poco distante con los ORMs a través de Sequelize, ¿recuerdas? Finalmente llegó la hora de otorgarle el protagonismo que se merece y entrar en las profundidades de este universo. Revisemos de qué se trata. Un ORM (del inglés, Object-Relational Mapping) es un modelo de programación que consiste básicamente en establecer correspondencias entre campos de una base de datos a otra, a esto se lo llama mapear. La primera es la base de datos que uses para almacenar tus datos, y la segunda es una base de datos virtual que se genera en el lenguaje de programación con el que estés programando tu aplicación, es decir con cual estés usando un ORM. Esto te permitirá, como ya te debes imaginar, convertir los datos de tus objetos en un formato accesible para guardar la información en una BD, generando una comunicación fluida entre nuestra aplicación y bases de datos no relacionales. De esta manera, se creará una base de datos virtual donde estarán vinculados los datos de nuestra aplicación.



En términos simples, lo que debes hacer es definir un objeto que tendrá los mismos atributos que campos tenga la tabla a representar, así cada atributo del objeto se corresponde con un campo de la tabla.



Además de mapear, los ORMs tienden a “liberarte” de la escritura de código SQL necesario para realizar las *queries*o consultas, y te permiten gestionar la persistencia de datos en la base de datos.

Existen muchas herramientas para cada lenguaje de programación. Debes encontrar la que más se adapte a tus necesidades y la que más aporte de la comunidad tenga. Ejemplos de estas pueden ser: [Doctrine](https://www.doctrine-project.org/projects/doctrine-orm/en/2.7/tutorials/getting-started.html) (PHP), [Hibernate](https://hibernate.org/orm/documentation/5.4/" \t "_blank) (Java) y [Sequelize](https://sequelize.org/master/manual/getting-started.html" \t "_blank) (Node JS).

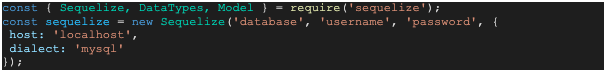
**Pero, ¿cómo funciona todo esto?**

Imagina que necesitas guardar datos de personas en una tabla que se llame “*Users*”, dicha tabla podría tener dos datos, como son el nombre y apellido. Estos campos podrían ser de tipo *varchar*, ya que representan cadenas de caracteres.

Para visualizar fácilmente cómo funciona vamos a realizar las operaciones CRUD (Create, Read, Update y Delete).

**1. Conexión**

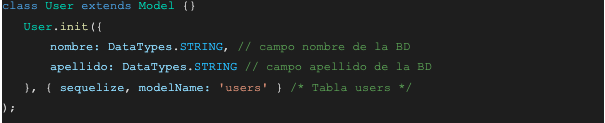
Antes de realizar cualquier operación debemos estar conectados a la DB con las credenciales necesarias para poder ejecutar las instrucciones. Luego, definiremos nuestra conexión en una variable llamada sequelize a través de la cual podremos ejecutar todos los métodos.



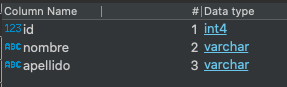
[Nota:](https://parrot.acamica.com/encuentros/26?group=DWBE-ONLINE-2" \t "_blank)

[Recuerda instalar Sequelize y mysql con NPM en tu sprint project.](https://parrot.acamica.com/encuentros/26?group=DWBE-ONLINE-2" \t "_blank)

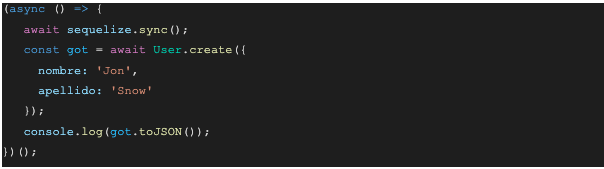
**2. Crear**



Con este ORM, puedes definir una clase User, con las claves nombre y apellido, definiéndolas como string, tal cual lo hiciste en la tabla.



Una vez tengas esto definido, tendrás la posibilidad de realizar diferentes operaciones, por ejemplo, insertar datos. Sequelize será el encargado de almacenar el nombre y apellido en su base de datos virtual (la DB mapeada) para luego guardarlo definitivamente en tu motor de base de datos.



Cuando quieras crear un usuario, solo basta con indicar el valor de cada clave, y el ORM se encargará de guardarlo en la tabla.



Como habrás notado, se pueden guardar datos usando un lenguaje de programación orientado a objetos en una base de datos sin la necesidad de recurrir a SQL. En otras palabras, no tendrías que escribir las sentencias de INSERT, DELETE o UPDATE.

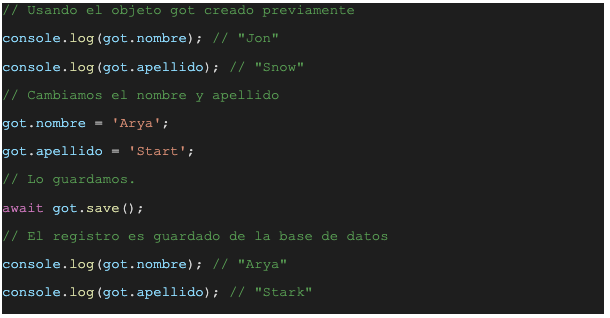
Entonces, Sequelize se encarga de mapear tu objeto *User*con las claves apellido y nombre desde su base de datos virtual hacia la base de datos que hayas definido. Lo que hace internamente el ORM es ejecutar un *query*como el que se presenta debajo, en la base de datos real:



Piensa esto: ¿cuántas líneas de código crees que te llevaría hacer la misma consulta por tu cuenta?

**3. Actualizar**

Actualizar tu base de datos es una de las operaciones más importantes, donde puedes modificar todos, o algunos, de los campos del usuario. Veamos cómo se puede realizar:



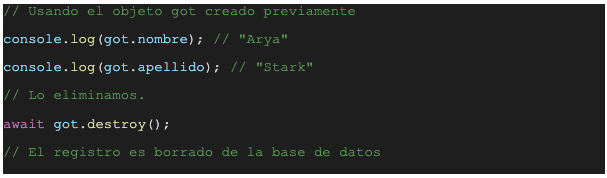
Como podrás observar, se cambia el valor de las claves nombre y apellido en el objeto, y al guardarlas se reflejan dichos cambios en la base de datos. El *query*entonces sería el siguiente:



Luego de haber guardado los cambios, el registro de en la base de datos se vería así:



**4. Eliminar**



En lenguaje SQL sería algo asi:



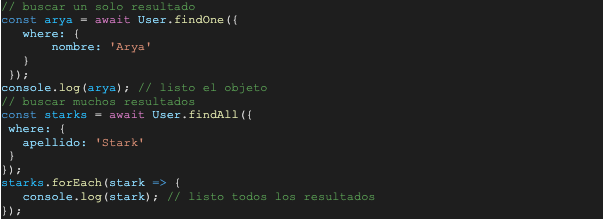
Y la tabla ahora quedará vacía.



**5. Buscar**

Cómo es habitual en cualquier aplicación, siempre es necesario buscar información, ¿te imaginas que cada vez que necesitemos un dato debamos traer todos los registros de una tabla? Tu aplicación tendría una performance muy pobre. Para evitar esta problemática, el ORM nos permite buscar información de manera certera, tal cual lo hacemos con WHERE en el lenguaje SQL.

Puedes hacerlo de la siguiente forma:



En SQL en cambio se vería así:

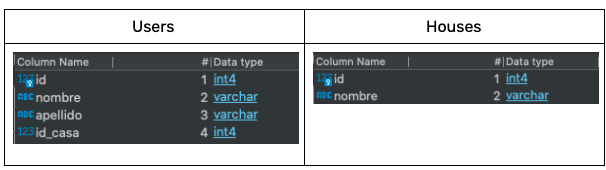


Esta es una de las combinaciones posibles de búsqueda. Puedes encontrar más combinaciones de búsqueda en [este sitio](https://sequelize.org/master/manual/model-querying-basics.html#applying-where-clauses) .

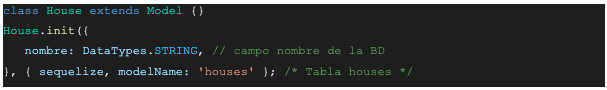
**¿Cómo nos relacionamos?**

Cómo hemos visto en la meeting anterior, en SQL tenemos la posibilidad de vincular 2 tablas a través de la cláusula join, mediante un campo en común dentro de ellas. Esto también puede hacerse en Sequelize, veamos cómo.

Primero vamos a modificar nuestra tabla *Users*y crear una nueva tabla para tener una relación. De esta manera, nuestro nuevo modelo de base de datos quedará así:



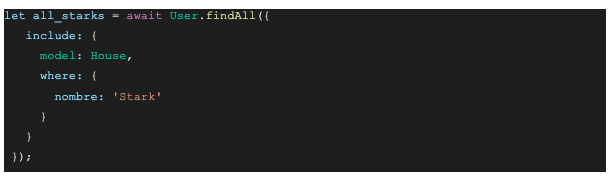
Luego debemos crear el modelo para la nueva tabla *Houses*.



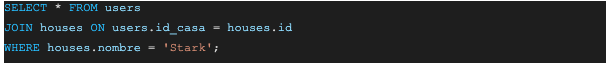
Además del modelo, se define la relación entre ambas tablas para poder usar el join.



Como ya tenemos la relación definida, podemos buscar todos los usuarios que tengan una relación con la tabla *Houses* donde el nombre de su casa sea igual a “Stark”.



En SQL se vería así:



**Ventajas y desventajas**

¿Vienes bien hasta aquí? Ya vimos algunas de las funcionalidades básicas de un ORM y su utilidad, pero como todo tienen tanto sus pros y contras. Veamos un poco más acerca de estas.

Una de las ventajas más importantes de un ORM, es que **no necesitas saber el lenguaje de cada base de datos** que vayas a utilizar, ya que esto se traduce inmediatamente y hace que sea más **fácil y rápido implementar** comunicaciones con bases de datos. Es decir, permite que puedas **centrarte únicamente en el desarrollo de tu aplicación.**

Otras ventajas son:

1. Facilita la migración de datos. Si algún día necesitas cambiar de base de datos, no es necesario reconstruir las consultas.
2. Código más legible y con menos líneas.
3. Evita escribir las consultas a mano.
4. Agrega una capa de seguridad, ya que tiene una capa independiente a los datos.

Pero, como ya sabes, siempre hay un lado oculto de las cosas. Algunas desventajas de utilizar un ORM son:

1. Se puede llegar a tener mal performance si no se usa debidamente.
2. No dispone de todas las funcionalidades de un [motor de base de datos](https://es.wikipedia.org/wiki/Sistema_de_gesti%C3%B3n_de_bases_de_datos) .

**Cierre**

Un ORM permite centrarte mejor en tu aplicación y, además, puedes concentrarte en diseñar un modelo de objetos y organizar toda la aplicación alrededor de este modelo, sin tener que preocuparte en cómo almacenarlos ni en pensar cuál base de datos debes utilizar. ¡Pongamos esto en práctica!