# Mongo II

**Solidificando los datos**

\*“Un comentario es un síntoma de no haber conseguido escribir un código claro.” Robert C.Martin — Ing. de Software\*

A estas alturas ya sabes qué es una [BD](https://es.wikipedia.org/wiki/Base_de_datos), que existen las bases de datos [SQL](https://es.wikipedia.org/wiki/SQL) y las [noSQL](https://es.wikipedia.org/wiki/NoSQL" \t "_blank), qué es un modelo, qué es un esquema y cómo se guardan y se obtienen datos de ellas. ¿Cuántos nuevos conceptos, no cierto? Respira...

A continuación explicaremos con más detalle nuevas funcionalidades sobre las bases de datos para profundizar en temas que hay que tener en cuenta al momento de administrar y trabajar con los datos de tu BD: las **transacciones** y los **documentos embebidos.**

Transacciones

Las transacciones son una funcionalidad de las bases de datos que te permitirán realizar acciones más complejas. Muchas veces nos encontramos en la necesidad de realizar múltiples instrucciones en un BD para dar como exitosa una operación (es decir, el paquete de instrucciones) que deseamos realizar. Imagina la siguiente situación, estás desarrollando un nuevo *[feature](https://www.javiergarzas.com/2012/09/metodologia-agil-fdd-2.html" \t "_blank)* para que un home banking tenga la posibilidad de transferir dinero de una cuenta a otra. En este ejemplo deberás realizar más de una instrucción para que tu operación pueda ser considerada exitosa.

Operación: El usuario A quiere transferir U$S 250 al Usuario B

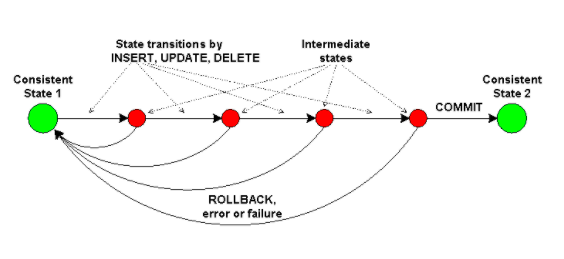
Instrucciones:

* ¿Usuario A cuenta con todas las condiciones para recibir dinero? (cuenta activa, alguna inhibición, problemas legales, deudas, etc)
* ¿Usuario A tiene fondos disponibles?
* ¿Usuario B existe?
* ¿Usuario B cuenta con todas las condiciones para recibir dinero? (cuenta activa, alguna inhibición, problemas legales, deudas, etc)
* Restar U$S 250 a Usuario A
* Sumar U$S 250 a Usuario B
* Actualizar saldo Usuario B
* Guardar un LOG de la operación

Como tal vez puedas inferir, cada una de las instrucciones que describimos es una ejecución en la base de datos. Existen diversas razones por las cuales la operación puede fallar, a simple vista la más sencilla puede ser que el Usuario A no tenga fondos. De ser así, todas las instrucciones que siguen dejan de tener sentido.

Ahora bien, imagina esta otra situación: todas las instrucciones vienen ejecutándose de manera correcta, tu código llega al momento donde debe restarle U$S 250 al Usuario A. Está instrucción se ejecuta con normalidad, pero acto seguido el servidor de base de datos deja de estar disponible (cualquiera sea el problema, se cae, se rompe físicamente, hay problemas de red, etc). Entonces, la siguiente operación de sumar U$S 250 al Usuario B no puede ejecutarse por los problemas del servidor. Aquí nos encontramos con un problema muy grande: hemos descontado dinero de una cuenta y nunca ha llegado a su destino.

¡Es ahora donde las transacciones vienen al rescate! En esta situación, todas las instrucciones deben ser exitosas para que realmente se graben en la base de datos, si alguna de ellas falla ninguna se guardará en tu DB, es decir, es como si nunca las hubieras ejecutado.



[Fuente](https://maxdb.sap.com/)

Entremos en un lenguaje más técnico: una [transacción](https://es.wikipedia.org/wiki/Transacci%C3%B3n_(inform%C3%A1tica)) es un conjunto de operaciones sobre la BD, donde todas las operaciones dentro de ella se validan o se cancelan como si fuera una única operación.

Para ser considerada como tal, una transacción debe cumplir con las propiedades [ACID](https://es.wikipedia.org/wiki/ACID), que aseguran que una operación sea exitosa:



[Fuente](https://www.marklogic.com/)

1. Atomicidad (**A**tomicity): debe ejecutarse en su totalidad o no ejecutarse; no puede quedar a medias.
2. Consistencia (**C**onsistency): siempre y cuando no atente la integridad de la BD.
3. Aislamiento (**I**solation): una transacción no debe afectar a otras.
4. Durabilidad (**D**urability): una vez realizada la transacción, ésta persistirá en la BD.

Todas las operaciones unitarias son tratadas internamente como transacciones por el motor de BD, pero si necesitas ejecutar un conjunto de ellas y quieres estar seguro de que todas van a persistir, es mejor indicarle que sean tratadas como una sola operación, usando la transacción.

En pseudocódigo lo puedes ver de la siguiente manera:

[start transaction]

[Instrucción 1]

[Instrucción 2]

[Instrucción 3]

[end transaction]

Todo el conjunto de instrucciones debe ser exitosa para que la transacción finalice de manera satisfactoria, y así los datos finalmente queden persistentes en la DB. Si alguna instrucción falla, todo vuelve al inicio (al punto del *start transaction*), cómo si ninguna instrucción hubiera sido ejecutada.

Ahora pasemos al código. Lo primero que tenemos que hacer para utilizar las transacciones en[MongoDB](https://es.wikipedia.org/wiki/MongoDB) es conocer los métodos que ofrecen.

Por un lado, tenemos métodos para gestionar la sesión:

* startSession(): las transacciones están asociadas a una sesión, es por ello que siempre necesitaremos abrir una sesión.
* endSession(): cuando cerremos la sesión, si hay alguna transacción ejecutándose, esta se abortará.

Por el otro lado, encontramos los métodos para controlar la transacción:

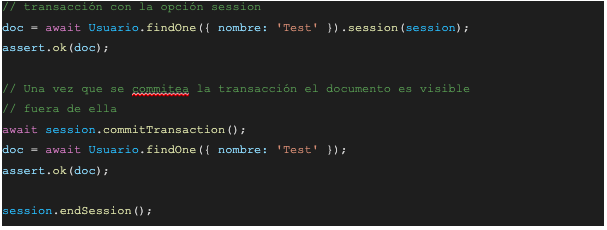
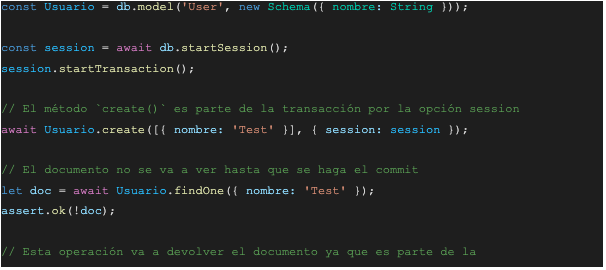
* Session.startTransaction(): este método nos sirve para empezar la transacción dentro de una sesión.
* Session.commitTransaction(): con este método confirmaremos los cambios que hayamos realizado dentro de la transacción.
* Session.abortTransaction(): aborta la ejecución de la transacción y, por lo tanto, no graba las modificaciones hechas a lo largo de la transacción.

**Métodos para la transacciones**

¿Vienes bien hasta aquí? Es mucha teoría, ¿verdad? No te preocupes, llegó la hora de poner toda esta información en práctica, para ver cómo funcionan efectivamente estos métodos que te comentamos más arriba para controlar transacciones. Empecemos viendo un ejemplo con Commit.

**Commit**

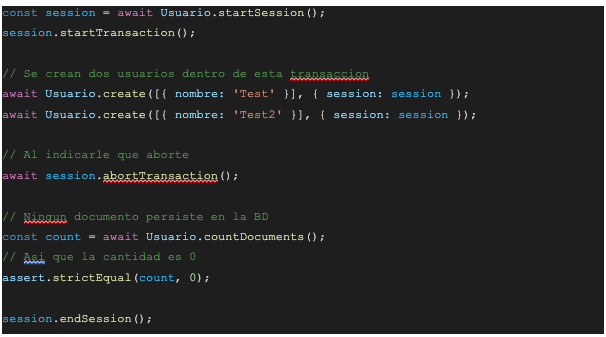
A continuación, te mostraremos cómo funciona el [commit](https://es.wikipedia.org/wiki/Commit" \t "_blank) de la transacción usando [Mongoose](https://mongoosejs.com/docs/index.html" \t "_blank):



Ten en cuenta, si no ocurrió algún error durante la ejecución de la transacción y se llegó a la instrucción de commit, esto quiere decir que todas las operaciones dentro de la transacción fueron persistidas en la BD.

**Abort**

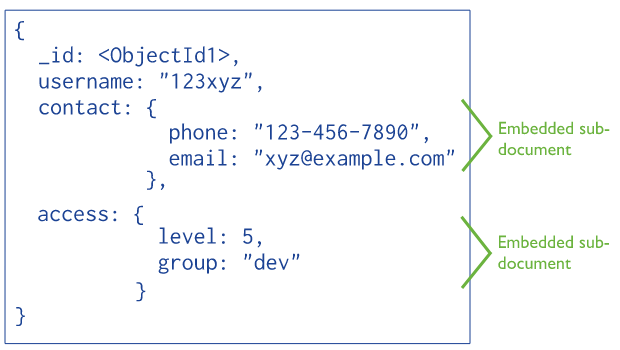
El caso presentado debajo muestra qué pasaría si ejecuta el abort en [Mongoose](https://mongoosejs.com/docs/index.html" \t "_blank):



Cuando la ejecución de la transacción llega a la instrucción abort, lo que hace es deshacer todos los cambios hechos hasta el momento dentro de ella, por lo que la BD queda inalterada.

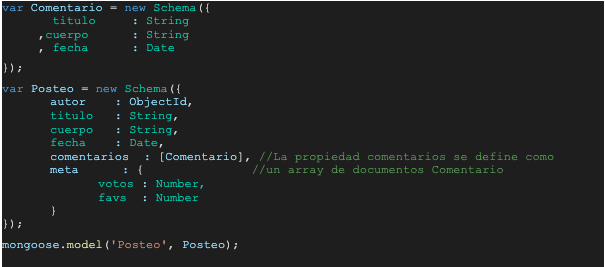
**Documentos Embebidos**

Como ya sabes, los datos en las Bases de Datos NoSQL pueden ser almacenados en documentos, como por ejemplo en MongoDB. Cada documento está definido con un esquema, que establece las propiedades y los tipos de valores que van a ser guardados en ellos.



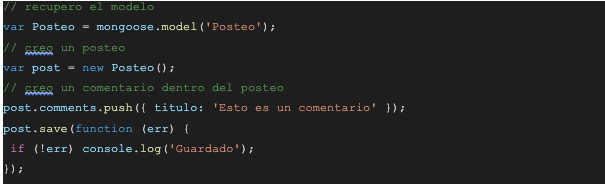
[Fuente](https://docs.mongodb.com/)

A veces puede ser necesario almacenar documentos complejos, que pueden contener entidades que tienen sus propias propiedades, ya sea porque se necesita información extra o para agilizar las búsquedas de información dentro de ellas. Así nacen los documentos embebidos. Esto es básicamente un documento dentro de otro, donde por ejemplo, una propiedad dentro de un documento A, puede ser declarada como un documento B. Es decir, básicamente se trata de documentos con esquemas propios que forman parte de otros documentos. Para verlo de manera más sencilla, te mostramos un ejemplo usando Mongoose:



En este caso se define que un posteo tiene un array de comentarios. A continuación, te mostraremos algunas operaciones básicas que puedes realizar sobre estos documentos.

* Cómo agregar un documento embebido a un array



* Cómo eliminar un documento embebido



* Cómo encontrar un documento embebido



**Cierre**

Habrás notado que usar documentos embebidos ayuda a definir correctamente entidades complejas y a agilizar la gestión sobre ellas. Al tener todos los datos contenidos en un mismo documento se hace más rápido y fácil buscar información sobre él, ya que al no contar con las relaciones, como en las BD SQL, es más simple almacenar documentos dentro de otros para no tener que armar una lógica extensa y tener que buscarlos en distintos modelos.

Además, como has visto, el uso de transacciones permite que puedas insertar, por ejemplo, muchos comentarios en un posteo y asegurarte que todos se hayan guardado correctamente, o si alguno de ellos falló, que el posteo no haya perdido su integridad usando una transacción.

El uso de estas características hace que tu modelo de datos y aplicación sea mucho más robusta y segura al momento de administrar tus datos. Ten paciencia, practica, y de a poco ¡todo irá funcionando!v