Pre-localización

Cuando sabemos que un documento o colección va a albergar un tipo de datos que va a crecer en gran medida y además controlamos cómo crecerá podemos diseñar la BBDD para que sea lo más eficiente posible, este caso se da por ejemplo en las series temporales.

Imaginemos que guardamos la temperatura en tiempo real segundo a segundo de una planta de sensores, la primera intención suele ser guardar un documento para cada medida de cada sensor:

```
{       timestamp:ISODate("2016-10-10T23:06:37.000Z"),
        Valor:24.56
},{     timestamp:ISODate("2016-10-10T23:06:38.000Z"),
        Valor:23.98
}
```

Sin embargo, esto no es nada eficiente, podríamos almacenar todas las medidas que se realizar durante un minuto:

```
{
        timestamp:ISODate("2016-10-10T23:06:38.000Z"),
        Valores:{
0:23.98,
                1:23.87,
                …,
                59:21.98
}
```

Además sabemos que las operaciones de modificación son más eficientes que las de inserción, puesto que si un campo ya existía en disco, solo hay que cambiar su valor, no es necesario relocalizar el documento si este es muy grande.

Si quisiéramos guardar toda una hora de datos, podríamos pensar en algo así:

```
{
        timestamp:ISODate("2016-10-10T23:06:38.000Z"),
        Valores:{
0:23.98,
                1:23.87,
                …,
                3598:22.14,
```

```
                3599:21.98
}
```

Sin embargo, podríamos hacerlo de manera más eficiente aún:

```
{
        timestamp:ISODate("2016-10-10T23:06:38.000Z"),
        Valores:{
0:{0:23.98,1:23.87,...,58:23.84,59:23.81},
                1:{0:23.74,1:23.71,...,58:23.62,59:23.49},
                …
                59:{0:23.24,1:23.20,...,58:23.06,59:23.00},
}
```

https://www.mongodb.com/blog/post/ schema-design-for-time-series-data-in-mongodb

Two phase commit

Las operaciones en mongodb son atómicas a nivel de documento, perdemos la ventaja de las operaciones de las base de datos SQL, por eso vamos a ver un ejemplo de cómo realizar transacciones en mongoDB.

Supongamos que queremos realizar un transferencia de dinero entre dos cuentas A y B y asegurarnos de que la operación se ha realizado de manera correcta.

Deberemos crear dos colecciones, una con información de las cuentas y otra con información acerca de las transacciones, los pasos a realizar son:

1. Insertar un nuevo documento con la transacción de la cuenta A a la B que incluya el estado “inicial”, la hora de la inserción y la cantidad de la transferencia.

2. Modificamos el estado de la transacción a pendiente y actualizamos la hora de la última modificación.

3. Modificamos la cuenta A con -cantidad y una transacción pendiente en el array de transacciones pendientes

4. Modificamos la cuenta B con +cantidad y una transacción pendiente en el array de transacciones pendientes

5. Modificamos la transacción a estado aplicada así como la hora de última actualización

6. Eliminamos de ambas cuentas la transacción pendiente del array de transacciones pendientes

7. Modificamos la transacción a un estado finalizada

```
> use twoPhaseCommits
switched to db thoPhaseCommits
> db.cuentas.insert({_id:"A", balance:600, transaccionesPendientes:[]})
WriteResult({ "nInserted" : 1 })
> db.cuentas.insert({_id:"B", balance:900, transaccionesPendientes:[]})
WriteResult({ "nInserted" : 1 })
> db.transacciones.insert({_id:1, fuente:"A", destino:"B", cantidad:100, estado:"inicial", ulti
maModificacion: new Date()})
WriteResult({ "nInserted" : 1 })
> var t = db.transacciones.find({estado:"inicial"})
> t
{ "_id" : 1, "fuente" : "A", "destino" : "B", "cantidad" : 100, "estado" : "inicial", "ultimaMo
dificacion" : ISODate("2017-02-13T21:46:14.341Z") }

> db.transacciones.update({_id:1, estado:"inicial"},{$set:{estado:"pendiente"},$currentDate:{ul
timaModificacion:true}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> var t = db.transacciones.find({estado:"pendiente"})
> t
{ "_id" : 1, "fuente" : "A", "destino" : "B", "cantidad" : 100, "estado" : "pendiente", "ultima
Modificacion" : ISODate("2017-02-13T21:48:35.153Z") }
> db.cuentas.update({_id:"A", transaccionesPendientes:{$ne:1}},{$inc:{balance:-100},$push:{tran
saccionesPendientes:1}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.cuentas.update({_id:"B", transaccionesPendientes:{$ne:1}},{$inc:{balance:100},$push:{trans
accionesPendientes:1}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.transacciones.update({_id:1, estado:"pendiente"},{$set:{estado:"hecha"},$currentDate:{ulti
maModificacion:true}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.cuentas.update({_id:"A", transaccionesPendientes:1},{$pull:{transaccionesPendientes:1}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.cuentas.update({_id:"B", transaccionesPendientes:1},{$pull:{transaccionesPendientes:1}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.transacciones.update({_id:1, estado:"hecha"},{$set:{estado:"finalizada"},$currentDate:{ult
imaModificacion:true}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.cuentas.find()
{ "_id" : "A", "balance" : 500, "transaccionesPendientes" : [ ] }
{ "_id" : "B", "balance" : 1000, "transaccionesPendientes" : [ ] }
> db.transacciones.find()
{ "_id" : 1, "fuente" : "A", "destino" : "B", "cantidad" : 100, "estado" : "finalizada", "ultim
aModificacion" : ISODate("2017-02-13T21:53:09.805Z") }
>
```

https://docs.mongodb.com/manual/tutorial/perform-two-phase-commits/