Base de Datos No SQL

Diagrama de modelo de datos NoSQL

A diferencia de SQL, que tiene diagramas de DER(Diagrama Entidad Relación) y el Modelo Relacional, NoSQL no tiene nombres ni restricciones para los diagramas de modelado de datos. La razón obvia son las reglas laxas de NoSQL sobre las relaciones, que tienen como objetivo que el desarrollador comience con requisitos mínimos.
En el modelado NoSQL, tendremos algunas formas de diseñar nuestra estructura de datos NoSQL, las posibilidades son:

- Diagrama UML (ER)
- Mapa mental
- Documento y Colecciones

Para cada modelo debemos plantear la estrategia a utilizar.

Ejemplo de enunciado del problema

Tomemos un enunciado de problema y dibujemos nuestro diagrama de modelo de datos en torno a él en pasos:

"*Hay una tienda de regalos llena de **artículos** de regalo . Varios **cajeros** realizan sus funciones en diferentes momentos del día y **venden** estos artículos a los clientes. Los clientes no están registrados en el sistema, pero se consulta su nombre y se incluye en el recibo.*"

En otras palabras, necesitamos crear un inventario básico de regalos y un sistema de pago a la luz de la información anterior. Por lo tanto, comenzamos por identificar entidades obvias e implícitas.

Las entidades

- **Artículo**
- **Producto** : no se menciona, pero es necesario distinguir entre, por ejemplo, el perfume Hugo Boss (producto) y 10 botellas del mismo (artículos).
- **Cajero**
- **Transacción** : una palabra más informativa que “vender” o “comprar”

No es necesario el cliente porque solo necesitamos su nombre y no tenemos más información. No es necesario un recibo porque todos los datos para imprimir en el recibo están disponibles en las transacciones.

La marca podría ser una entidad propia si queremos almacenar algo más que su nombre.

Listado de posibles consultas

Durante el proceso de modelado, siempre es útil planificar con anticipación las operaciones de lectura y escritura, para poder mantenerlas eficientes y recuperar información significativa de nuestros datos.

1. Lectura de datos

Los requisitos cambian con el tiempo y NoSQL está pensado para adaptarse a eso, pero debes pensar un poco en el mínimo inicial que puedes especificar desde el principio. En este caso, será:

- Clasificar las **marcas** y **productos** con mejor rendimiento dentro de un período de tiempo determinado
- Identificar los **artículos** que necesitan reposición
- **Desglose de transacciones** por cualquier período de tiempo: por hora, por día, por semana, por mes, por año (es posible que necesitemos dibujar y comparar gráficos para obtener información)
- Enumerar **los cajeros** según su antigüedad
- Un cliente ha perdido el ticket, pero sabe qué **productos** compró y cuándo. Consultando las **transacciones** , se imprime un duplicado del ticket
- Obtener todos **los productos** que cuesten menos de $50 (un cliente puede consultar al cajero)

2. Escritura de datos

En esta parte, deberíamos centrarnos principalmente en cómo se almacenan las referencias dada la cantidad de datos en expansión y la frecuencia de las actualizaciones. Analizar las “Relaciones” 1 a 1, 1 a muchos o muchos a muchos.

Tips:

Datos en constante expansión

Si la cantidad de documentos aumenta constantemente, nunca los coloque ni sus referencias en otro documento. Por ejemplo, la cantidad de **elementos** seguirá aumentando, por lo que no podemos incrustar ni agregar sus identificadores como referencia dentro de **un producto** , ya que eventualmente se volverá demasiado grande y se quedará sin el espacio máximo asignado por documento, sin mencionar que será lento de obtener. En cambio, deberíamos mantener la referencia del producto en cada elemento.

2. Escritura de datos

En esta parte, deberíamos centrarnos principalmente en cómo se almacenan las referencias dada la cantidad de datos en expansión y la frecuencia de las actualizaciones. Analizar las “Relaciones” 1 a 1, 1 a muchos o muchos a muchos.

Tips:

Datos en constante expansión

Si la cantidad de documentos aumenta constantemente, nunca los coloque ni sus referencias en otro documento. Por ejemplo, la cantidad de **elementos** seguirá aumentando, por lo que no podemos incrustar ni agregar sus identificadores como referencia dentro de **un producto** , ya que eventualmente se volverá demasiado grande y se quedará sin el espacio máximo asignado por documento, sin mencionar que será lento de obtener. En cambio, deberíamos mantener la referencia del producto en cada elemento.

2. Escritura de datos

En esta parte, deberíamos centrarnos principalmente en cómo se almacenan las referencias dada la cantidad de datos en expansión y la frecuencia de las actualizaciones. Analizar las “Relaciones” 1 a 1, 1 a muchos o muchos a muchos.

Tips:

Adiciones y actualizaciones frecuentes

Si algunos documentos requieren muchas adiciones, actualizaciones o eliminaciones, no los incorpore ni coloque su referencia en un solo lugar, porque podría resultar en:

Ralentizar las operaciones de la base de datos

Posibles datos inconsistentes, si las operaciones no se realizaron de forma atómica (es decir, se obtuvo un documento, se modificó en el lado del cliente y luego se actualizó. Entre la obtención y la actualización, es posible que otra persona haya actualizado el documento y ahora tiene datos obsoletos que su llamada de actualización conservará).

2. Escritura de datos

En esta parte, deberíamos centrarnos principalmente en cómo se almacenan las referencias dada la cantidad de datos en expansión y la frecuencia de las actualizaciones. Analizar las “Relaciones” 1 a 1, 1 a muchos o muchos a muchos.

Tips:

Adiciones y actualizaciones frecuentes

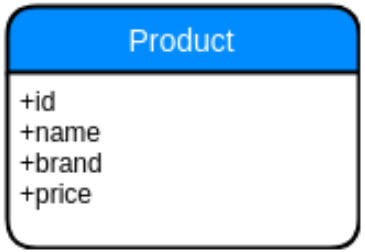
Por ejemplo : si insertas **comentarios** y **me gusta** de **los usuarios** en una **publicación** , la publicación recibe miles de comentarios y me gusta en poco tiempo. Muchos también actualizarían sus comentarios, lo que significa que la publicación deberá actualizarse miles de veces en un intervalo corto. Pero si **los comentarios** y **me gusta** tuvieran sus propias colecciones, cada comentario se habría agregado y actualizado por separado y de manera mucho más eficiente.

3. Modelado

Modelemos todas las entidades con diagramas, su relación con otras y documentos de ejemplo. Las colecciones de MongoDB se modelan como entidades y los documentos como json.

3. Modelado

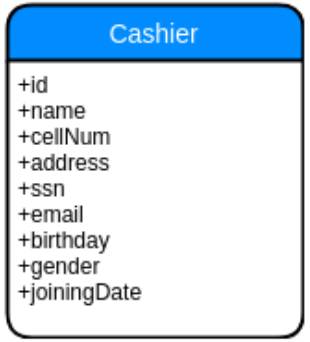
Producto




```
{
    id: 1,
    name: "Dark Blue Cologne",
    brand: "Hugo Boss",
    price: 22.46
}
```

3. Modelado

Cajero




```
{
    id: 1,
    name: "John Smith",
    cellNum: "4565456789",
    address: "Broadway Street house no. 12, Brooklyn, New York, US",
    ssn: "011-72-XXXX"
    .
    .
    .
}
```

3. Modelado

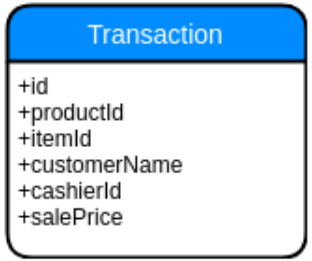
El producto tiene muchos elementos

En vista del razonamiento anterior, no hemos incluido los elementos incrustados ni sus referencias en una matriz dentro del documento del producto .
En su lugar, los guardamos productId como referencia en los elementos.

```
//products
{
    id: 1,
    name: "Dark Blue Cologne",
    brand: "Hugo Boss",
    price: 22.46
}

//items
{
    id: 1,
    productId: 1,
    barcode: "0705632441947"
}

{
    id: 2,
    productId: 1,
    barcode: "0705632441948"
}

{
    id: 3,
    productId: 1,
    barcode: "0705632441949"
}
```

3. Modelado

Transacción



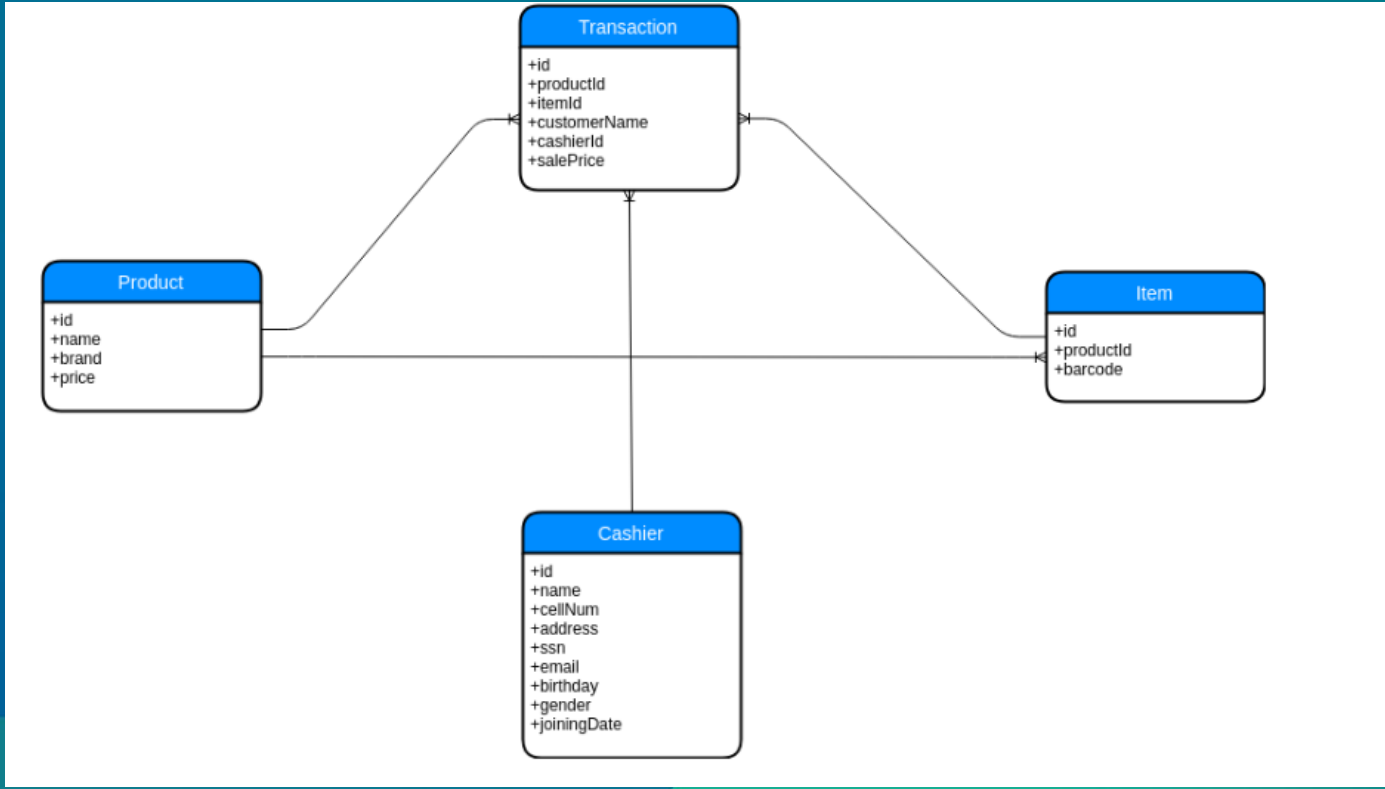
La transacción es una colección asociativa necesaria para resolver el siguiente problema de muchos a muchos:
Un cajero puede vender una cantidad ilimitada de productos (el cajero John vende 15 juegos de colonia Hugo's Dark Blue)
Un artículo de producto puede ser vendido por muchos cajeros (Hugo's Dark Blue Cologne es vendido por los cajeros John, Tony, Malinda, George y otros 2)
salePrice: Es necesario saber a qué precio se vendió el artículo del producto

3. Modelado

Puede resultar tentador modelar la transacción en un escenario del mundo real en el que un cliente compra varios artículos a la vez, que se muestran en el recibo como tal. Sin embargo, la transacción debe representar solo una venta. Para comprar varios artículos, digamos 10 a la vez, se deben crear 10 documentos de transacción.

Puede crear otra colección para agrupar varias transacciones o simplemente colocar un campo de identificador único (por ejemplo: transactionId) en todas las transacciones realizadas colectivamente a la vez.

3. Modelado


Transaction
+id
+productId
+itemId
+customerName
+cashierId
+salePrice
Product
+id
+name
+brand
+price
Item
+id
+productId
+barcode
Cashier
+id
+name
+cellNum
+address
+ssn
+email
+birthday
+gender
+joiningDate

3. Modelado

Esta relación también se puede describir como *El producto* tiene muchas *transacciones*, *El cajero* tiene muchas *transacciones*, *El artículo* tiene muchas *transacciones*.

" *El producto* tiene muchas *transacciones*" es completamente opcional y depende de los requisitos. Por ejemplo, cuando necesita generar informes anuales de ventas y, por lo tanto, necesita conocer los productos en datos agregados, es más fácil tener una referencia de producto en el documento de transacción que obtenerla a través del documento de artículo (que requerirá una operación de unión si está disponible en la base de datos o varias consultas).

3. Modelado

```
//products
{
    id: 1,
    name: "Dark Blue Cologne",
    brand: "Hugo Boss",
    price: 22.46
}

//items
{
    id: 1,
    productId: 1,
    barcode: "0705632441947"
}

{
    id: 2,
    productId: 1,
    barcode: "0705632441948"
}

{
    id: 3,
    productId: 1,
    barcode: "0705632441949"
}
```

```
// cashiers
{
    id: 1,
    name: "John Smith",
    cellNum: "4565456789",
    address: "Broadway Street house no. 12, Brooklyn, New York, US",
    ssn: "011-72-XXXX"
    .
    .
    .
}

{
    id: 2,
    name: "Malinda Johnson",
    cellNum: "4565456689",
    address: "Harold Street house no. 133, Brooklyn, New York, US",
    ssn: "011-73-XXXX"
    .
    .
    .
}
```

```
//transactions
{
    id: 1,
    productId: 1,
    itemId: 1,
    customerName: "Daniel Witz",
    cashierId: 1,
    salePrice: 22 //salePrice to provide discount option
}

{
    id: 1,
    productId: 1,
    itemId: 2,
    customerName: "Glenn McDowell",
    cashierId: 1,
    salePrice: 22
}

{
    id: 1,
    productId: 1,
    itemId: 3,
    customerName: "Robert Williams",
    cashierId: 2,
    salePrice: 22
}
```

Colecciones separadas en relación uno a uno; caso de especialización y generalización

Cuándo y por qué se debe utilizar la referencia para vincular documentos en una relación 1-1.

Para la relación uno a uno, por lo general, no resulta evidente por qué se necesita una colección separada en lugar de incrustar todo en un solo documento.

Colecciones separadas en relación uno a uno; caso de especialización y generalización

Ejemplo de universidad

Modelando las entidades surgen algunas entidades obvias, y los atributos mínimos requeridos para cada uno (sin incluir campos obvios como id, createDate, updateDate, etc.)

Alumno

```
firstName
lastName
dob
email
password
education
batch
CGPA
enrolDate
```

Profesor

```
firstName
lastName
dob
email
password
degrees
experience
bio
joinDate
```

Guardia

```
firstName
lastName
dob
email
password
weapon
experience
joinDate
```

Colecciones separadas en relación uno a uno; caso de especialización y generalización

Como modelar

- **Todas las entidades tienen sus propios modelos/colecciones**

Cada una de las entidades mencionadas anteriormente tiene sus propios modelos.
A primera vista, parece una buena opción, pero si se analiza más a fondo, se revela un problema. Cuando la lógica de la aplicación se escribe en torno a este diseño, y especialmente cuando tenemos un único punto de entrada al sistema (la misma página de inicio de sesión/API sin parámetros para identificar el rol) para todos los tipos de usuarios, es necesario que busquemos en cada una de las cinco colecciones una combinación de correo electrónico y contraseña para averiguar qué usuario ha iniciado sesión (y quizás llevarlo a su propio panel de control).

Colecciones separadas en relación uno a uno; caso de especialización y generalización

- **Todas las entidades en una sola colección**

Muchos de los campos enumerados anteriormente son comunes a todas las entidades. Eso incluye *email* y *password*, que son credenciales para iniciar sesión. Esta característica común nos ofrece una solución fácil al problema anterior: podemos fusionar todas las entidades actuales en una colección, por ejemplo Person o User, y mantener otro campo *userType* (o *userRole*) que indique el tipo de usuario.

Colecciones separadas en relación uno a uno; caso de especialización y generalización

- **Todas las entidades en una sola colección**

Nos ahorra el problema de identificar el tipo de usuario y la dificultad de inicio de sesión al fusionar campos comunes como *firstName*, *lastName*, *email*, *password*, etc. en una sola colección. Al mismo tiempo, dificulta la gestión de aplicaciones. Porque la mayoría de los otros campos son exclusivos de tipos de usuarios específicos, por ejemplo, solo student tiene *batch* y *enrolDate*; solo professor tiene *degrees*; solo guard tiene *weapon*; y así sucesivamente. Eso implica mucho seguimiento y gestión.
Mongodb no almacena ningún campo como nulo si no se proporciona pero aún así nuestra colección es muy difícil de escalar a medida que se incorporan más roles de usuario al sistema.

Usuario

```
firstName
lastName
dob
email
password
education
batch
CGPA
certificates
degrees
experience
joinDate
enrolDate
vocationalTraining
weapon
bio
userType
```

Colecciones separadas en relación uno a uno; caso de especialización y generalización

- **Una colección general y especializada**

La mejor manera, es la relación uno a uno en forma dividida, es decir, utilizar dos colecciones y vincularlas con una referencia. En este caso, *Usuario* es la colección generalizada que contiene campos comunes, mientras que todos los demás tipos de usuarios son colecciones especializadas que solo contienen los campos relevantes.

En las relaciones 1-1, la elección de la colección a la que se debe mantener la referencia es arbitraria, ya que tanto el documento referenciado como el documento al que se hace referencia son únicos y, en términos de rendimiento, no hay mucha diferencia en mantener la referencia en ambos lados.

Colocamos el campo *user* en todas las colecciones especializadas para mantener la referencia *del usuario* (ID del usuario).

Colecciones separadas en relación uno a uno; caso de especialización y generalización

- **Una colección general y especializada**

Usuario

```
firstName
lastName
dob
email
password
education
userType
```

Alumno

```
education
batch
CGPA
enrolDate
user
```

Profesor

```
degrees
experience
bio
joinDate
user
```

Guardia

```
weapon
experience
user
```

Con esta división, resulta muy fácil agregar más roles. La lógica de inicio de sesión también requiere poco o ningún cambio con cada incorporación.

Colecciones separadas en relación uno a uno; caso de especialización y generalización

Conclusión

La mayoría de las veces, en la fase de diseño y modelado de datos NoSQL, no parecemos encontrar un caso relevante de separación de colecciones para casos uno a uno. Por lo general, tampoco es necesario, ya que los casos comunes se atienden bien mediante la incrustación del documento. Pero en este caso, analizamos un ejemplo en el que es mejor dividir las colecciones (en sus formas especializadas y generalizadas) y usar referencias para vincularlas, para una mejor gestión de la aplicación y una fácil escalabilidad.

.

Colecciones separadas en relaciones uno a muchos; casos limitados y no limitados

Ejemplos del mundo real de relaciones de uno a muchos, tanto limitadas como no limitadas, en las que es posible (o no) aplicar diferentes formas de modelado.

Uno a muchos limitadas

Tomemos como ejemplo un *restaurante* con muchos *trabajadores* . Si analizamos esta relación, es obvio que está limitada por un límite superior. Los trabajadores que trabajan en un restaurante siempre estarán limitados en número, por ejemplo, 30. Pase lo que pase, nunca se van a descontrolar, ni siquiera si se incrementan en miles, y estamos absolutamente seguros de ello. Por lo tanto, podemos decir que la relación está **limitada** .

Colecciones separadas en relaciones uno a muchos; casos limitados y no limitados

Uno a muchos limitados

Otro ejemplo *un avión* tiene muchas *partes*: millones. Como las partes del avión no aumentan, la relación sigue siendo limitada. Sin embargo, el documento del avión será demasiado grande para ser manejable (o incluso permitido por la base de datos).
Por lo tanto, podemos decir que la relación está **limitada**, pero es inmanejable.
Las opciones de modelado son dos: ***matriz de referencias en un lado***; o ***referencia del documento de pertenencia en la otra relación***.

Colecciones separadas en relaciones uno a muchos; casos limitados y no limitados

Uno a muchos limitados

Matriz de referencias en un lado

Restaurante

```
{
    id: "restaurant1",
    .
    .
    .
    .
    workers: ["worker1", "worker2", "worker3"]
}
```

Trabajadores

```
{
    id: "worker1",
    .
    .
    .
}
{
    id: "worker2",
    .
    .
    .
}
{
    id: "worker3",
    .
    .
    .
}
```

Colecciones separadas en relaciones uno a muchos; casos limitados y no limitados

Uno a muchos limitados

Matriz de referencias en un lado

Avión

```
{
     id: "plane1",
     .
     .
     .
     parts: ["part1", "part2", "part3" ...]
}
```

Regiones

```
{
     id: "part1",
     .
     .
     .
}
{
     id: "part2",
     .
     .
     .
}
{
     id: "part3",
     .
     .
     .
}
```

Colecciones separadas en relaciones uno a muchos; casos limitados y no limitados

Uno a muchos limitados

Referencia del documento de pertenencia en la otra relación

Restaurante

```
{
    id: "restaurant1",
    .
    .
    .
    .
}
```

Trabajadores

```
{
    id: "worker1",
    .
    .
    .
    restaurant: "restaurant1"
}
```

```
{
    id: "worker2",
    .
    .
    .
    restaurant: "restaurant1"
}
```

```
{
    id: "worker3",
    .
    .
    .
    restaurant: "restaurant1"
}
```

Colecciones separadas en relaciones uno a muchos; casos limitados y no limitados

Uno a muchos limitados

Referencia del documento de pertenencia en la otra relación

Avión

```
{
     id: "plane1",
     .
     .
     .
}
```

Regiones

```
{
     id: "part1",
     .
     .
     .
     plane: "plane1"
}
```

```
{
     id: "part2",
     .
     .
     .
     plane: "plane1"
}
```

```
{
     id: "part3",
     .
     .
     .
     plane: "plane1"
}
```

Colecciones separadas en relaciones uno a muchos; casos limitados y no limitados

De uno a muchos sin límites

Solo funciona la ***Referencia del documento de pertenencia en la otra relación***.

Un ejemplo sería que una *publicación* tiene muchos *visitantes* . Es decir, registramos todas y cada una de las visitas a nuestra publicación y cierta información básica como *la dirección IP* , *el navegador* , *el tipo de dispositivo* (computadora de escritorio, tableta, dispositivo móvil), etc. Incluso si esperamos unos pocos miles de visitas por publicación como máximo, no estamos seguros de que una publicación se vuelva viral y atraiga más de un millón de visitas. Por lo tanto, tenemos un caso de relación de uno a muchos **sin límites** .

Colecciones separadas en relaciones uno a muchos; casos limitados y no limitados

De uno a muchos sin límites

Post

```
{
    id: "post1",
    .
    .
    .
}
```

Visitor

```
{
    id: "visitor1",
    .
    .
    .
    post: "post1"
}
```

```
{
    id: "visitor2",
    .
    .
    .
    post: "post1"
}
```

```
{
    id: "visitor3",
    .
    .
    .
    post: "post1"
}
```

Colecciones separadas en relaciones uno a muchos; casos limitados y no limitados

De uno a muchos sin límites

Conclusión

Para resumir:

- Si está limitado y es pequeño, la referencia se puede colocar en cualquier lado, es decir se puede colocar la ***matriz de referencias de cualquier lado***.
- Si está limitado pero es demasiado grande para mantener la ***matriz de referencias de un lado***, se realiza la ***Referencia del documento de pertenencia en la otra relación.***
- Si no está limitado, siempre se realiza la ***Referencia del documento de pertenencia en la otra relación***

Colecciones separadas en relaciones de muchos a muchos; casos limitados y no limitados

Enfoque correcto para modelar relaciones de muchos a muchos y uso de colecciones asociativas

De muchos a muchos

Ejemplo práctico de "Un taxi es asignado a muchos conductores“ y "Un conductor es asignado a muchos taxis“

Colecciones separadas en relaciones de muchos a muchos; casos limitados y no limitados

De muchos a muchos

Esta relación está limitada en ambos extremos, como máximo, un conductor puede ser asignado a un número limitado de taxis y, de manera similar, un taxi no puede ser conducido por más de un puñado de conductores.

Debido a dicha limitación, estamos seguros de que la asignación en cualquiera de los lados no puede ni siquiera superar los 10, y mucho menos miles o millones. Por lo tanto, este es el caso limitado.

Colecciones separadas en relaciones de muchos a muchos; casos limitados y no limitados

De muchos a muchos

Siempre podemos mantener una ***matriz de referencias de cualquier lado***

La ***colección asociativa*** también es posible en casos limitados, pero una nueva colección suele ser una exageración.

Colecciones separadas en relaciones de muchos a muchos; casos limitados y no limitados

De muchos a muchos

Taxis

```
{
    id: "cab1",
    .
    .
    .
    .
    drivers: ["driver1", "driver2", "driver3"]
}
```

```
{
    id: "cab2",
    .
    .
    .
    .
    drivers: ["driver1", "driver2"]
}
```

Conductores

```
{
    id: "driver1",
    .
    .
    .
}
```

```
{
    id: "driver2",
    .
    .
    .
}
```

```
{
    id: "driver3",
    .
    .
    .
}
```

Colecciones separadas en relaciones de muchos a muchos; casos limitados y no limitados

De muchos a muchos

Conductores

```
{
    id: "driver1",
    .
    .
    .
    .
    cabs: ["cab1", "cab2", "cab3"]
}
```

```
{
    id: "driver2",
    .
    .
    .
    .
    cabs: ["cab1", "cab2", "cab3"]
}
```

Taxis

```
{
    id: "cab1",
    .
    .
    .
}
```

```
{
    id: "cab2",
    .
    .
    .
}
```

```
{
    id: "cab3",
    .
    .
    .
}
```

Colecciones separadas en relaciones de muchos a muchos; casos limitados y no limitados

De muchos a muchos sin límites

Ejemplo: *un usuario* puede dar me gusta a muchas *publicaciones* y *muchas publicaciones* pueden dar me gusta a muchos *usuarios* . Observe la infinitud de esto. Un usuario puede dar me gusta a millones de publicaciones y, de manera similar, una publicación puede dar me gusta a millones de usuarios.

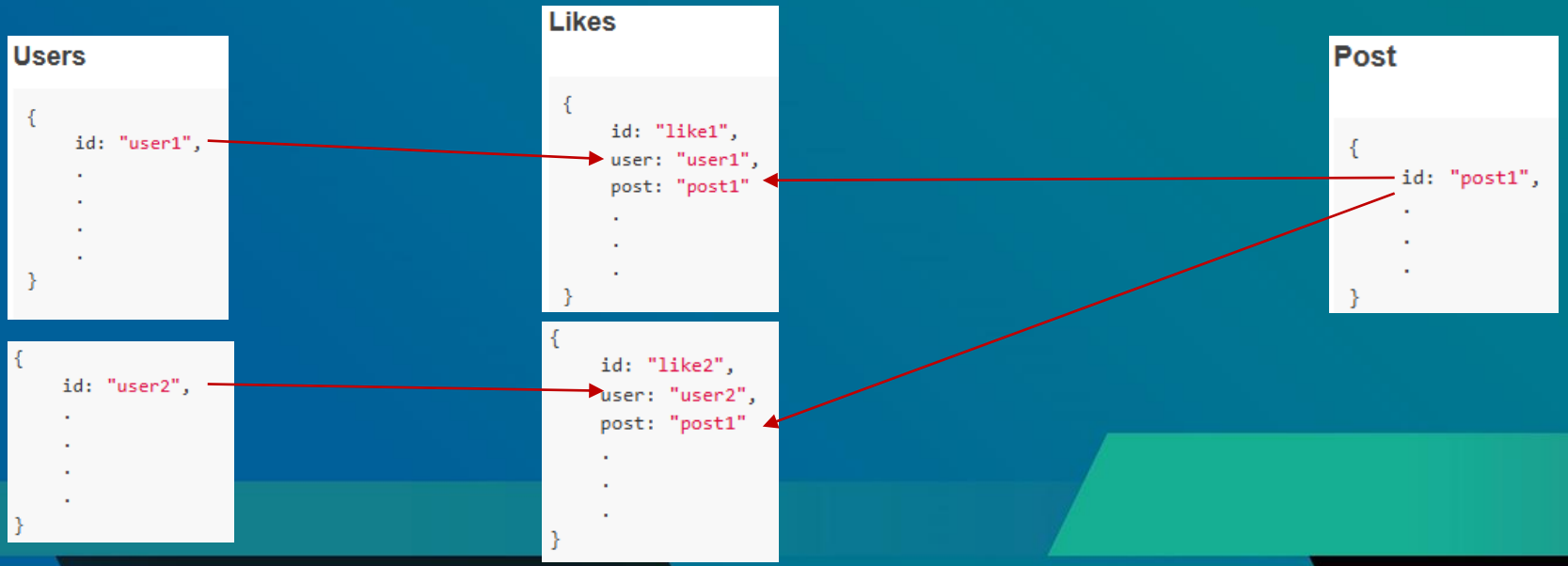
Colecciones separadas en relaciones de muchos a muchos; casos limitados y no limitados

De muchos a muchos sin límites

En estos casos, la ***matriz de referencia*** no se puede colocar en ninguno de los lados de la relación (usuario o publicación) ya que las referencias cada vez mayores superarán el límite de tamaño de un documento en poco tiempo.
La única solución factible es una ***colección asociativa*** intermedia, que mantenga las referencias de ambos lados.

Colecciones separadas en relaciones de muchos a muchos; casos limitados y no limitados

De muchos a muchos sin límites



Colecciones separadas en relaciones de muchos a muchos; casos limitados y no limitados

De muchos a muchos sin límites

Conclusión

- Si está limitados y es pequeño, se puede colocar una ***matriz de referencias*** en cualquier lado
- Si está limitado pero es demasiado grande para mantener la ***matriz de referencias*** o si no está limitado, la referencia debe colocarse en un ***documento asociativo*** entre los dos.