Base de Datos No SQL

Modelado de datos NoSQL: 1 a 1, 1 a muchos, muchos a muchos

Relaciones

A diferencia de SQL, el modelado de datos NoSQL permite múltiples formas de modelar relaciones de 1 a 1, de 1 a muchos y de muchos a muchos. No impone reglas ni favorece un diseño en particular. La única restricción son los requisitos de la aplicación.

1 a 1

Tomemos la colección *de clientes* y enumeremos todas las formas posibles de relación uno a uno con *la dirección*, es decir, *el cliente* tiene *la dirección* o *la dirección* pertenece al *cliente*:

Valores primitivos (caso trivial)

Los valores primitivos son, por defecto, 1 a 1. Forman el documento básico. *El cliente* tiene *id*, *firstName*, *lastName*, *registration* y *address*. (O *id* pertenece al *cliente*, *address* pertenece al *cliente*, etc.)

1 a 1

{ id: 1, firstName: "John",
lastName: "Smith",
registered: true,
address: "Broadway Street house no. 12, Brooklyn, New York, US" }

Ventajas

La información completa está contenida dentro del documento y se puede recuperar fácilmente sin tener que buscar en otros documentos.

Contras

La replicación se producirá si un valor de campo es compartido por otros documentos, por ejemplo, address: "Broadway Street house no. 12, Brooklyn, New York, US"en varios clientes

Documento incrustado/subdocumento

Se prefiere cuando el documento incrustado es breve y único o mayoritariamente único, es decir, cada cliente tiene una dirección única, excepto unos pocos que comparten residencia, en cuyo caso se producirá una replicación de la dirección, pero esto debería ser una rareza y se puede ignorar.

```
{       id: 1, firstName: "John",
        lastName: "Smith",
        registered: true,
        address: { houseNum: 12,
                   street: "Broadway Street",
                   state: "New York",
                   city: "Brooklyn",
                   country: "US" }      }
```

Documento incrustado/subdocumento

Ventajas

La información completa está contenida dentro del documento y se puede recuperar fácilmente sin tener que buscar en otros documentos.

Contras

La replicación cuando el documento incrustado es exactamente la misma en otros documentos

Colección referenciada

A veces, el documento incrustado, aunque único, se vuelve muy grande. Se puede mantener incrustado, pero existe la opción de crear una colección separada y hacer referencia a ella. De esta manera, *la dirección* no forma parte del documento *del cliente*, sino que solo está vinculada a él a través del campo de referencia (en este caso dirección).

```
{
    id: 1,
    firstName: "John",
    lastName: "Smith",
    registered: true,
    address: "address1"

}

{
    id: "address1"
    houseNum: 12,
    street: "Broadway Street",
    state: "New York",
    city: "Brooklyn",
    country: "US"
    .
    .
    .
}
```

Colección referenciada

Generalmente, el lado importante conserva la referencia, que en este caso es *Cliente*.
Pero también podemos vincular las mismas colecciones al revés, colocando la referencia de *Cliente* en *Dirección*.

```
{
    id: "client1",
    firstName: "John",
    lastName: "Smith",
    registered: true

}

{
    id: "address1"
    houseNum: 12,
    street: "Broadway Street",
    state: "New York",
    city: "Brooklyn",
    country: "US"
    client: "client1"
    .
    .
    .
}
```

Colección referenciada

Ventajas

El tamaño del documento se mantiene bajo control colocando cierta información en un documento separado y conservando solo su referencia.

Contras

Para obtener información completa, es necesario obtener dos documentos en lugar de uno.

1 a muchos

Continuando con el ejemplo de Cliente y Dirección, ampliémoslo para que un cliente pueda tener una o más direcciones, es decir, *el Cliente* tiene muchas *Direcciones* o muchas *Direcciones* pertenecen al *Cliente.*

Matriz de valores primitivos

Continuando con el ejemplo de Cliente y Dirección, ampliémoslo para que un cliente pueda tener una o más direcciones, es decir, *el Cliente* tiene muchas *Direcciones* o muchas *Direcciones* pertenecen al *Cliente.*

```
{       id: 1,
        firstName: "John",
        lastName: "Smith",
        registered: true,
        address: ["Broadway Street house no. 12, Brooklyn, New York, US", "Harold Street house
                no. 77, Brooklyn, New York, US"] }
```

Matriz de valores primitivos

Ventajas

La información completa se encuentra dentro del documento

Contras

La replicación se producirá si un valor de campo es compartido por otros documentos

Matriz de documentos incrustados/subdocumentos

```
{
    id: 1,
    firstName: "John",
    lastName: "Smith",
    registered: true,
    address: [{
        houseNum: 12,
        street: "Broadway Street",
        state: "New York",
        city: "Brooklyn",
        country: "US"
    },
    {
        houseNum: 77,
        street: "Harold Street",
        state: "New York",
        city: "Brooklyn",
        country: "US"
    }]

}
```

Matriz de documentos incrustados/subdocumentos

Ventajas

La información completa se encuentra dentro del documento

Contras

La replicación cuando el documento incrustado es exactamente la misma en otros documentos

El tamaño del documento se volverá inmanejable cuando la matriz incrustada sea demasiado grande.

Se requieren demasiadas actualizaciones para agregar/actualizar/eliminar subdocumentos cada vez mayores.

Demasiadas operaciones de actualización simultáneas darán como resultado el acceso al mismo documento, lo que ralentizará la operación y, en el peor de los casos, hará que los datos sean inconsistentes (si las operaciones no se manejan de forma atómica, es decir, el documento cambia entre el intervalo de búsqueda y actualización).

Matriz de referencias

```
{
    id: 1,
    firstName: "John",
    lastName: "Smith",
    registered: true,
    addresses: ["address1", "address2"]
}

{
    id: "address1"
    houseNum: 12,
    street: "Broadway Street",
    state: "New York",
    city: "Brooklyn",
    country: "US"
    .
    .
    .
}

{
    id: "address2"
    houseNum: 77,
    street: "Harold Street",
    state: "New York",
    city: "Brooklyn",
    country: "US"
    .
    .
    .
}
```

Matriz de referencias

Ventajas

El tamaño del documento se mantiene bajo control al mantener únicamente las referencias

Solo necesitamos tener un documento para tener toda la información para su uso posterior. Ejemplo: se recupera el documento del cliente. Para obtener direcciones, todo lo que necesitamos son las referencias disponibles en lugar de buscar en toda la colección de *direcciones* para consultar los documentos que pertenecen al *cliente*.

Matriz de referencias

Contras

Incluso con solo referencias en la matriz, el tamaño del documento podría salirse de control si no se planifica adecuadamente. Por ejemplo, *Video* tiene más de un millón de *Me gusta* (y sigue aumentando) y todas sus referencias se guardan en el documento *Video* como matriz

Se requieren demasiadas actualizaciones en el documento que contiene las referencias debido a las referencias constantes y en constante aumento/actualización.

Aunque es mejor que los documentos incrustados, también puede enfrentar algunos problemas si se requieren tantas operaciones simultáneas para actualizar las referencias, especialmente cuando las operaciones no se realizan de forma atómica.

Referencia en el lado “Pertenece a”

```
{
    id: "client1",
    firstName: "John",
    lastName: "Smith",
    registered: true,
}

{
    id: "address1",
    houseNum: 12,
    street: "Broadway Street",
    state: "New York",
    city: "Brooklyn",
    country: "US",
    client: "client1"
    .
    .
    .
}

{
    id: "address2",
    houseNum: 77,
    street: "Harold Street",
    state: "New York",
    city: "Brooklyn",
    country: "US",
    client: "client1"
    .
    .
    .
}
```

Referencia en el lado “Pertenece a”

Ventajas

El documento de “un” lado no necesita saber sus pertenencias y por lo tanto no necesita actualizaciones frecuentes ni gran tamaño.

Contras

Dado que el documento no tiene referencia de sus pertenencias, se requiere una búsqueda en toda la colección para recuperarlas cuando sea necesario (la indexación resuelve este problema)

Si se elimina un documento, se deben eliminar/actualizar todos sus elementos o serán datos obsoletos e inútiles, por ejemplo, *una publicación* eliminada con miles de *comentarios* donde *el comentario* tiene el ID *de la publicación*.

Muchos a muchos

Matriz de valores primitivos

{ id: 1,
firstName: "John",
lastName: "Smith",
registered: true,
address: ["Broadway Street house no. 12, Brooklyn, New York, US",
"Harold Street house no. 77, Brooklyn, New York, US"] }
{ id: 2,
firstName: "George",
lastName: "Smith",
registered: true,
address: ["Broadway Street house no. 12, Brooklyn, New York, US"] }

Matriz de valores primitivos

Ventajas

Posiblemente se salven dos colecciones y con ellas muchas referencias y su gestión.

Contras

Replicación garantizada, lo que significa múltiples operaciones de adición, actualización y eliminación al cambiar un elemento de la matriz.

Matriz de documentos integrados

```
{
    id: 1,
    firstName: "John",
    lastName: "Smith",
    registered: true,
    address: [{
        houseNum: 12,
        street: "Broadway Street",
        state: "New York",
        city: "Brooklyn",
        country: "US"
    },
    {
        houseNum: 77,
        street: "Harold Street",
        state: "New York",
        city: "Brooklyn",
        country: "US"
    }]

}

{
    id: 2,
    firstName: "William",
    lastName: "Smith",
    registered: true,
    address: [{
        houseNum: 12,
        street: "Broadway Street",
        state: "New York",
        city: "Brooklyn",
        country: "US"
    }]

}
```

Matriz de documentos integrados

Ventajas

Se guardaron colecciones adicionales y gestión de referencias

Contras

Múltiples operaciones de adición/actualización/eliminación en el cambio de elementos incrustados en una matriz

Replicación de documentos

Matriz de referencias

{ id: 1,
firstName: "John",
lastName: "Smith",
registered: true,
addresses: ["address1", "address2"] }
{ id: 2,
firstName: "George",
lastName: "Smith",
registered: true,
addresses: ["address1"] }

{ id: "address1"
houseNum: 12,
street: "Broadway Street",
state: "New York",
city: "Brooklyn",
country: "US" . . . }

{ id: "address2"
houseNum: 77,
street: "Harold Street",
state: "New York",
city: "Brooklyn",
country: "US" . . . }

Matriz de referencias

Contras

No es escalable. Cuando cualquiera de los lados de la estrategia de muchos a muchos crece demasiado, este enfoque es difícil de gestionar. Por ejemplo, una *publicación* puede tener "Me gusta" para millones de *usuarios* y un *usuario* puede tener "Me gusta" para una cantidad ilimitada de *publicaciones*, por lo que mantener referencias de matrices de usuarios a los que les gusta en una publicación o publicaciones a las que les gusta en un usuario es inviable.

Colección Conectiva/Asociativa In Between

```
//Client
{
    id: "client1",
    firstName: "John",
    lastName: "Smith",
    registered: true,
    addresses: ["address1", "address2"]
}

{
    id: "client2",
    firstName: "George",
    lastName: "Smith",
    registered: true,
    addresses: ["address1"]
}
```

```
//ClientAddress
{
    id: "clientAddress1"
    client: "client1"
    address: "address1"
}

{
    id: "clientAddress2"
    client: "client1"
    address: "address2"
}

{
    id: "clientAddress1"
    client: "client2"
    address: "address1"
}
```

```
//Address
{
    id: "address1"
    houseNum: 12,
    street: "Broadway Street",
    state: "New York",
    city: "Brooklyn",
    country: "US"
    .
    .
    .
}

{
    id: "address2"
    houseNum: 77,
    street: "Harold Street",
    state: "New York",
    city: "Brooklyn",
    country: "US"
    .
    .
    .
}
```

Colección Conectiva/Asociativa In Between

Ventajas

El único diseño escalable para una relación ilimitada de muchos a muchos, como que a un *usuario* le pueden gustar un número ilimitado de *publicaciones* y una *publicación* puede recibir me gusta de un número ilimitado de *usuarios.*

Contras

Recopilación adicional y, por lo tanto, operaciones adicionales de obtención/adición/actualización/eliminación