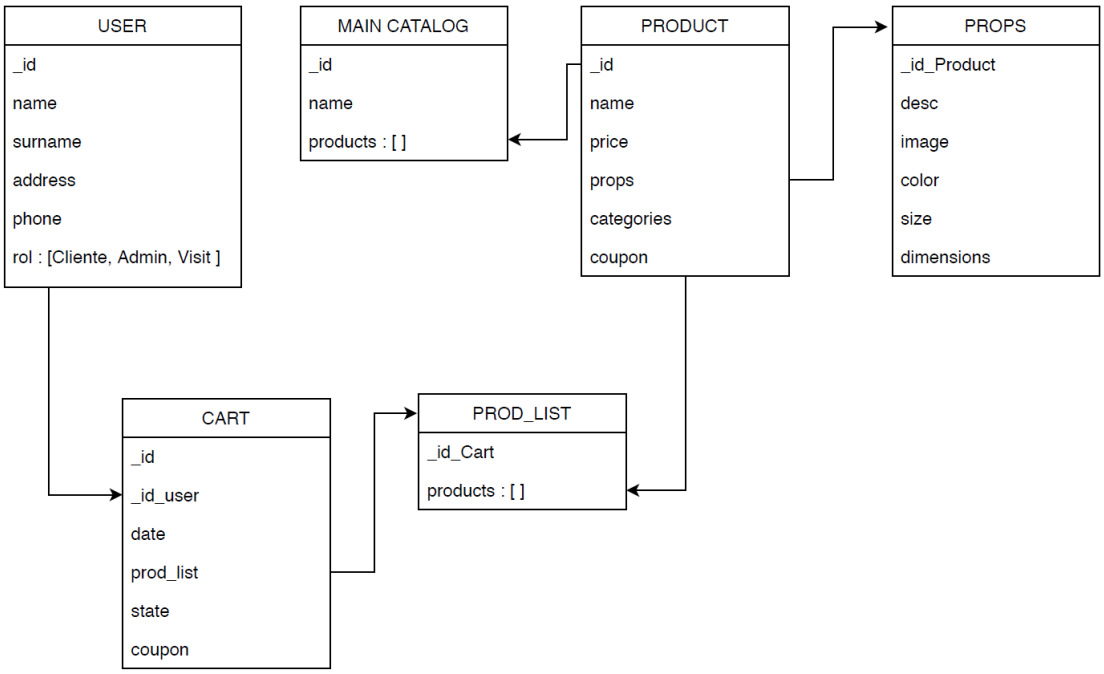
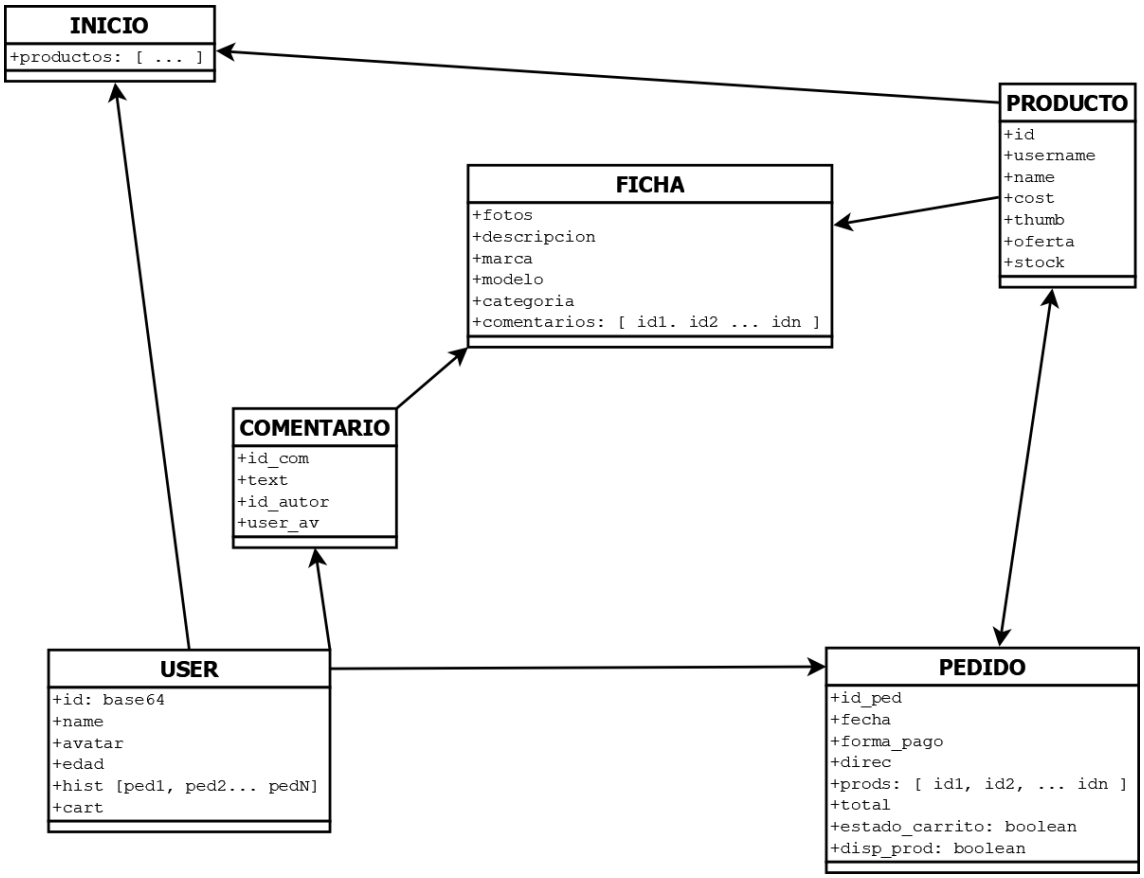
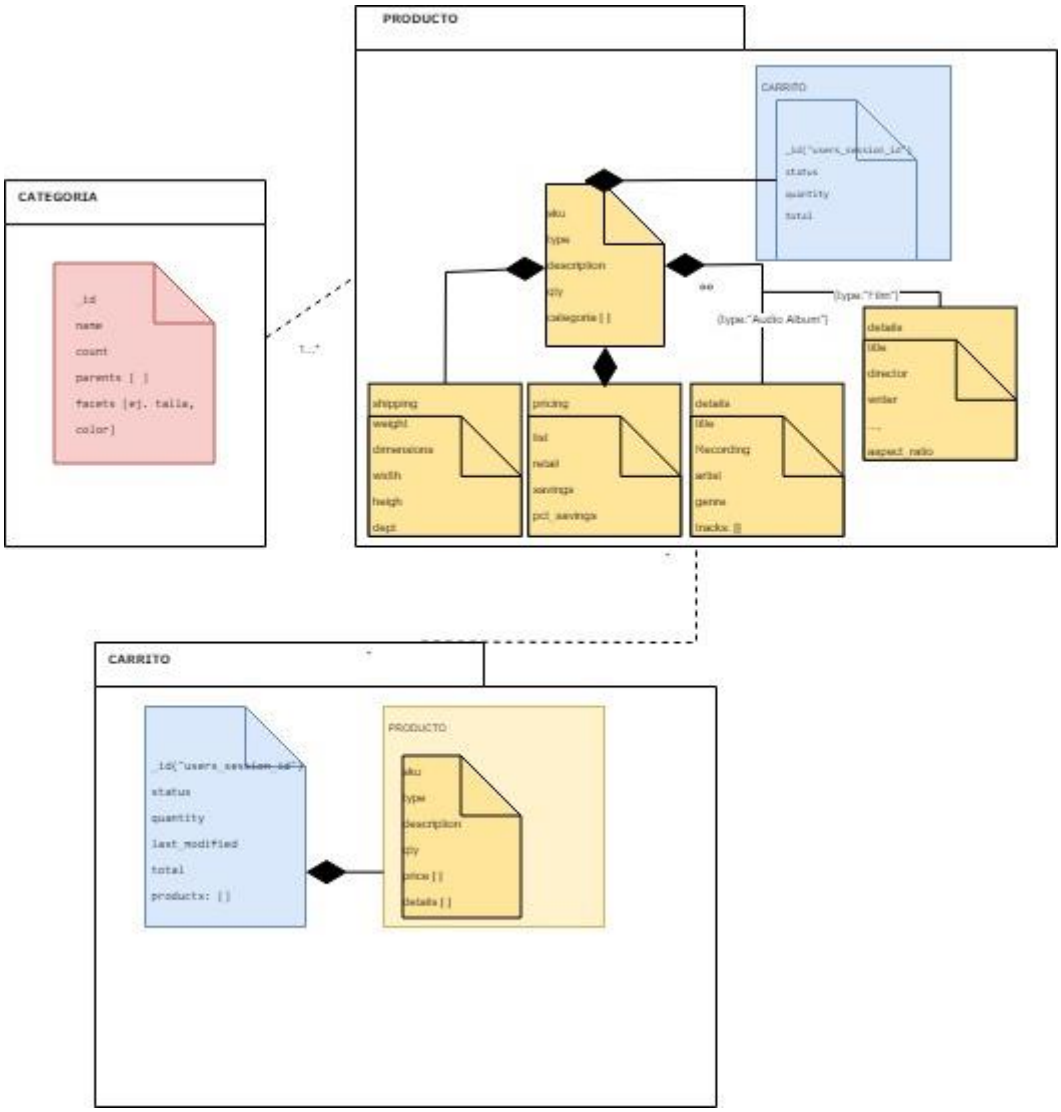
Catálogo de Productos

Para gestionar un sistema ecommerce, la primera cosa que se necesita es un **catálogo de productos**. Debe tener capacidad para almacenar diferentes tipos de objetos con diferente conjunto de atributos. Este tipo de colecciones de datos encajan bastante bien en el model de bd de MongoDB.




INICIO
+productos: [...]
PRODUCTO
+id
+username
+name
+cost
+thumb
+oferta
+stock
FICHA
+fotos
+descripcion
+marca
+modelo
+categoria
+comentarios: [id1. id2 ... idn]
COMENTARIO
+id_com
+text
+id_autor
+user_av
USER
+id: base64
+name
+avatar
+edad
+hist [ped1, ped2... pedN]
+cart
PEDIDO
+id_ped
+fecha
+forma_pago
+direc
+prods: [id1, id2, ... idn]
+total
+estado_carrito: boolean
+disp_prod: boolean



{
sku: "00e8da9b",
type: "Audio Album",
title: "A Love Supreme",
description: "by John Coltrane",
asin: "B0000A118M",
shipping: {
weight: 6,
dimensions: {
width: 10,
height: 10,
depth: 1
},
},
pricing: {
list: 1200,
retail: 1100,
savings: 100,
pct_savings: 8
},
details: {
title: "A Love Supreme [Original Recording Reissued]",
artist: "John Coltrane",
genre: ["Jazz", "General"],

```
...
tracks: [
"A Love Supreme, Part I: Acknowledgement",
"A Love Supreme, Part II: Resolution",
"A Love Supreme, Part III: Pursuance",
"A Love Supreme, Part IV: Psalm"
],
},
}

{
sku: "00e8da9d",
type: "Film",
...,
asin: "B000P0J0AQ",
shipping: { ... },
pricing: { ... },
details: {
title: "The Matrix",
director: [ "Andy Wachowski", "Larry Wachowski" ],
writer: [ "Andy Wachowski", "Larry Wachowski" ],
...,
aspect_ratio: "1.66:1"
},
}
```


Operaciones

1. ### Buscar productos ordenados por porcentaje de descuento en orden descendiente

La mayoría de las búsquedas será por tipo de producto, pero en algunas situaciones pueden realizarse búsquedas en un determinado rango de precio o porcentaje de descuento.

Búsqueda con un descuento mayor del 25%, requiere un índice en porcentaje de descuento (pct_savings)

2. ### Buscar álbumes por género y ordenar por año

Requiere crear un índice compuesto por género y fecha

3. ### Buscar películas basadas en el actor protagonista ordenadas por fechas

Requiere un índice por tipo, actor y fecha

4. ### Buscar películas con una palabra en el título

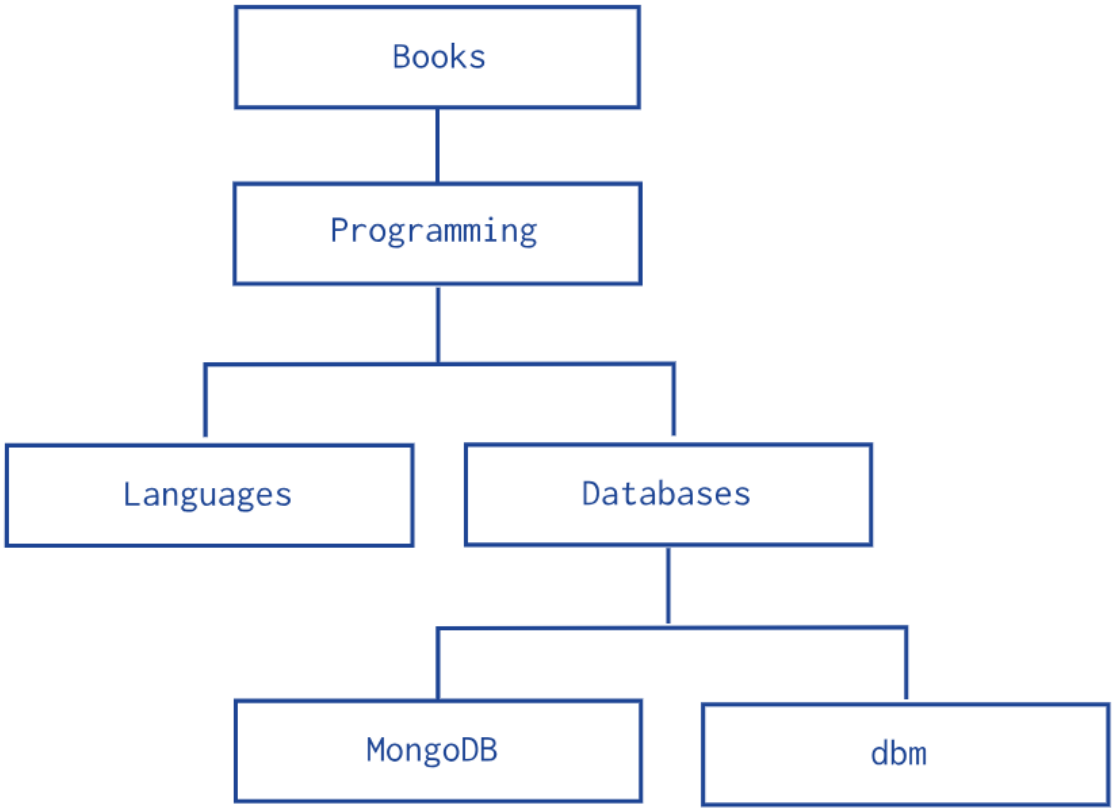
Requiere un índice tipo texto

5. Conclusion: Indéxalo todo!

En sistemas de ecommerce no se sabe por que campos buscará el usuario así que es mejor crear índices para todas las búsquedas posibles.

Jerarquía de Categorías

En MongoDB existen distintas maneras de representar un árbol de jerarquías:



1. Referencias de “parent”:

```
db.categories.insert( { _id: "MongoDB", parent: "Databases" } )
db.categories.insert( { _id: "dbm", parent: "Databases" } )
```

Se puede crear un índice sobre el campo parent

2. Referencias de “children”:

```
db.categories.insert( { _id: "MongoDB", children: [] } )
```

```
db.categories.insert( { _id: "dbm", children: [] } )
db.categories.insert( { _id: "Databases", children: [ "MongoDB", "dbm" ] } )
```

Las referencias por hijo es una solución aceptable para árboles que no requieren mantenimiento de subárboles. También para nodos con múltiples padres.

3. Referencias de arrays de ascendientes:

```
db.categories.insert( { _id: "MongoDB", ancestors: [ "Books", "Programming",
"Databases" ], parent: "Databases" } )
db.categories.insert( { _id: "dbm", ancestors: [ "Books", "Programming",
"Databases" ], parent: "Databases" } )
```

El *Array of Ancestors* es una solución rápida y eficiente para buscar descendientes y ancestros de un nodo, creado un índice para los campos de ascendientes. Es una buena elección para trabajar con subárboles.

4. Referencias de rutes materializadas

El patrón de rutas materializadas almacena cada nodo del árbol en un documen junto a un id que representa la ruta de nodos de los ascendientes. Requiere trabajar con strings y expresiones regulares, pero permite trabajar con rutas parciales.

```
db.categories.insert( { _id: "Books", path: null } )
db.categories.insert( { _id: "Programming", path: ",Books," } )
db.categories.insert( { _id: "Databases", path: ",Books,Programming," } )
db.categories.insert( { _id: "Languages", path: ",Books,Programming," } )
db.categories.insert( { _id: "MongoDB", path: ",Books,Programming,Databases," } )
db.categories.insert( { _id: "dbm", path: ",Books,Programming,Databases," } )
```

5. Referencias de grupos anidados

```
db.categories.insert( { _id: "Books", parent: 0, left: 1, right: 12 } )
db.categories.insert( { _id: "Programming", parent: "Books", left: 2, right: 11 } )
db.categories.insert( { _id: "Languages", parent: "Programming", left: 3, right: 4 } )
db.categories.insert( { _id: "Databases", parent: "Programming", left: 5, right: 10 } )
db.categories.insert( { _id: "MongoDB", parent: "Databases", left: 6, right: 7 } )
db.categories.insert( { _id: "dbm", parent: "Databases", left: 8, right: 9 } )
```

Operaciones

1. Leer y mostrar una categoría

La operación más básica es mostrar una jerarquía de categoría basad en un *slug*. Este tipo de query se utiliza en ecommerce para generar una lista de "hilos de Ariadna" (breadcrumbs) que se muestra al usuario mientras navega.
Requiere indexar el campo slug

2. Añadir una categoría a la jerarquía

Modificar la jerarquía añadiendo una nueva categoría

3. Cambiar los ascendientes de una categoría

Reorganizar las categorías cambiándolas de ascendientes

4. Renombrar una categoría

Consideraciones

Se necesitan controlar las transacciones mediante código.
El patrón de referencias *parent* es una solución simple pero requiere muchas queries para recuperar todo el árbol
El patrón de *child* es una solución adecuada cuando no es necesario realizar operaciones en los árboles de descendientes. Patrón adecuado para grafos donde un nodo puede tener múltiples padres
El array de ascendientes no tiene ventajas específicas salvo para requerir constantemente el path de los nodos
Los patrones se pueden mezclar.

Ejemplos de código:
https://github.com/Voronenko/Storing_TreeView_Structures_WithMongoDB

Gestión de inventario

La función de reservas es uno de los requerimientos básicos de un sistema ecommerce. Además de las capacidades básicas de llenar un carro de la compra y pagar, los clientes deben estar informados de condiciones de fuera de stock de los productos, para no permitirles añadir productos a menos que estén disponibles.

En un carro de la compra:

- Los clientes añaden y eliminan productos
- Cambian las cantidades
- Abandonan el carro en cualquier momento
- Si hay problemas durante o después de la reserva se requiere mantener o cancelar la orden.

Estados de un carrito:
active
El usuario está activo y añade o quita productos del carrito.

pending
El carrito está reservado pero el pago todavía no se ha capturado.
Los productos no se pueden añadir o quitar del carrito.

expiring
El carrito lleva inactivo demasiado tiempo y está bloqueado.
Los productos se devuelven al inventario como disponibles.

expired
El carro está inactivo y no está disponible.
El usuario debe crear un nuevo carrito.

Requiere dos colecciones: **product** y **cart**.

La colección producto contiene un documento para cada producto
Un usuario añade a su carrito una SKU (stock-keeping unit).
La forma más simple es usar un número SKU como _id y guardar un contador con la cantidad de cada producto.
Se añade un campo de detalles para mostrar al usuario.

```
{ _id: '00e8da9b', qty: 16, details: ... }
```

Se almacena en el esquema del producto una lista de carritos con un sku determinado. Éste se utilizará como un control de productos para el sistema. Es decir, en el caso que haya inconsistencia de datos entre producto y carrito "gana" la lista de productos. Se requiere un mecanismo de limpieza cuando existan estas inconsistencias.

```
{ _id: '00e8da9b',
qty: 16,
carted: [
{ qty: 1, cart_id: 42,
timestamp: ISODate("2012-03-09T20:55:36Z"), },
{ qty: 2, cart_id: 43,
timestamp: ISODate("2012-03-09T21:55:36Z") }
]
}
```

En este caso Hay 16 productos pero dos carros que todavía no han completado el checkout

La colección del carrito debe tener un _id, state, last_modified para gestionar la fecha de caducidad y la lista de productos y cantidades

```
{ _id: 42,
last_modified: ISODate("2012-03-09T20:55:36Z"),
status: 'active',
items: [
{ sku: '00e8da9b', qty: 1, details: {...} },
{ sku: '0ab42f88', qty: 4, details: {...} }
]
}
```

Los detalles del producto se copian al carrito para no tener que volver a cargar el producto original. También sirve para mantener el precio en el caso de que se actualicen los precios.

Operaciones

1. Añadir un producto al carrito

Un producto no se añade nunca al carrito a no ser que haya stock suficiente

Pasos:
1. Actualizar el carrito asegurando que está activo y añadiendo una línea de producto.
2. Actualizar el inventario, decrementando el stock disponible, *solo si hay stock suficiente*.

3. Si la actualización del stock falla por falta de inventario se deshace la operación y se lanza una excepción al usuario.
Para esta operación no se requieren índices adicionales, todo va por _id

2. Modificar la cantidad del carrito

1. Actualizar carrito (asumiendo que hay stock suficiente).
2. Actualizar colección de productos *Si hay stock suficiente*
3. Deshacer actualización del carrito si no hay stock suficiente y lanzar una excepción

3. Final de la operación (reserva y pago)

Requiere capturar los detalles de pago y actualizar los productos del carrito una vez que se ha hecho el pago

1. Bloquear el carrito pasándolo a estado= pending.
2. Recoger el pago del carrito. Si esto falla, desbloquear el carro y volverlo a estado active.
3. Pasar el estado del carrito a complete.
4. Eliminar todas las referencias a este carrito desde la colección de productos

4. Devolver al stock los carritos expirados

1. Buscar todos los carritos más antiguos de un umbral determinado y bloquearlos para expiring.
2. Para carro "expiring", devolver todos los productos al stock disponible.
3. Una vez que se ha actualizado el producto poner el carrito a expired.

Requiere índice compuesto: status, last_modified

5. Gestión de errores

Se requieren limpiezas periódicas para eliminar inconsistencias de inventario

1. Find all the expiring carted items
2. Find all the carted items that matched
3. First Pass: Find any carts that are active and refresh the carted items
4. Second Pass: All the carted items left in the dict need to now be returned to inventory

Requiere un índice sobre la fecha de modificación