



# **Bootcamp: Arquiteto(a) de Dados**

## **Documento Arquitetural**

Autor: Ricardo Rubens Copini

Data: 20 de Janeiro de 2025



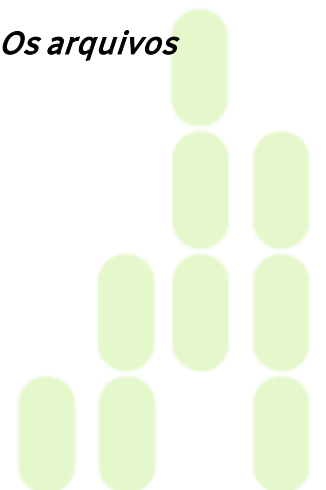
## Contexto da Arquitetura

Foi nos apresentado o contexto da loja “Amazonas”, empresa que atua no mercado digital oferecendo uma vasta gama de produtos. Tendo seu objetivo principal se tornar a maior loja de comércio eletrônico, de A a Z do Brasil.

Diante disto é esperado um volume considerável de dados e, portanto, seu sistema gerenciador de produtos e pedidos precisará ser desenvolvido para suportar escalabilidade para atender não só a questão do volume de dados, mas como também variações entre baixa e alta demanda de pedidos.

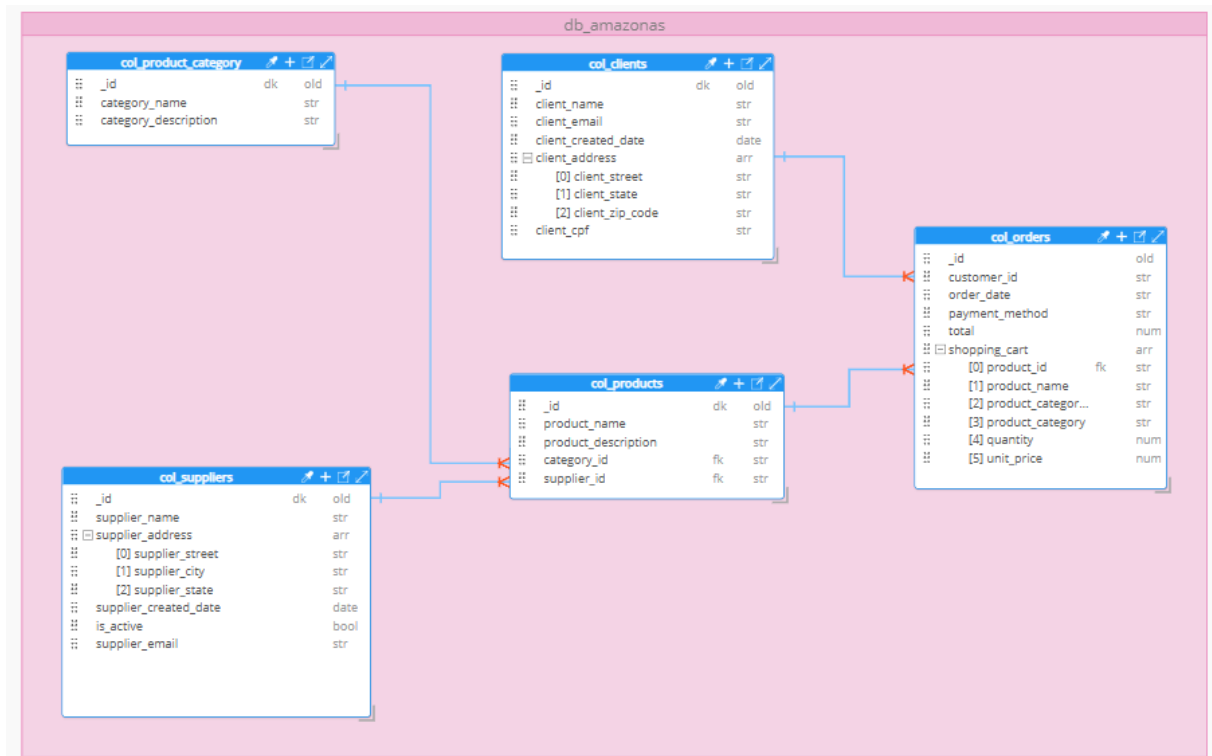
Para garantir que esta demanda seja atendida de maneira eficiente será utilizada uma abordagem de modelagem Não-Relacional NoSQL em MongoDB, acomodando todas as informações relacionadas à operação sem inferir um *schema* como também técnicas de *sharding* para que o sistema possa escalar horizontalmente sem ter perdas de performance.

*Professor, para este desafio eu tentei criar o banco usando Docker-Compose e consegui criar uma instância do MongoDB com um script de inicialização que carrega dados para as collections. Porém parei na parte das réplicas pois ao adicionar novos serviços do MongoDB eu precisaria usar um arquivo de chave para autenticação entre os serviços e não usar apenas usuário e senha admin/admin. Os arquivos estão no meu GitHub neste [link](#).*



# 1. Modelagem de Dados Não-Relacional

## Modelo Lógico (NoSQL)



### Coleção de Clientes (*col\_clients*)

col_clients		
::	_id	old dk
::	client_name	str
::	client_email	str
::	client_created_date	date
::	client_address	arr
::	[0] client_street	str
::	[1] client_state	str
::	[2] client_zip_code	str
::	client_cpf	str

A coleção de clientes (*col\_clients*) será composta dos dados de identificação do cliente como também uma coleção que determina sua localização de modo que seja possível planejar estratégias logísticas conforme a empresa se destaca no mercado. Os nomes dos campos, seus respectivos formatos e obrigatoriedade são definidos como:

Campo	Formato	Obrigatoriedade
_id	Object ID	Y
Client_name	String	Y
Client_email	String	Y
Client_created_date	Timestamp	Y
Client_address	Array	Y
Client_address.Client_street	String	N
Client_address.Client_state	String	N
Client_address.Client_zip_code	String	Y
Client_cpf	String	Y

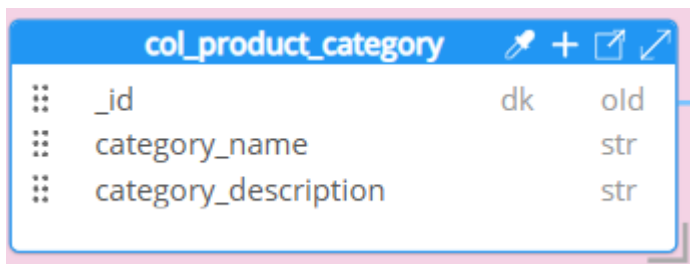
### Coleção de Produtos (*col\_products*)

col_products			
::	_id	dk	old
::	product_name		str
::	product_description		str
::	category_id	fk	str
::	supplier_id	fk	str

A coleção de produtos irá conter informações a nível de produto além de identificadores de categoria e fornecedores. Quando combinadas com informações da coleção de pedidos, possibilitam análises de desempenho por categoria como também identificar quais fornecedores são mais relevantes para o negócio. Os nomes dos campos, seus respectivos formatos e obrigatoriedade são definidos como:

Campo	Formato	Obrigatoriedade
_id	Object ID	Y
Product_name	String	Y
Product_description	String	N
Category_id	String (FK)	Y
Supplier_id	String (FK)	Y

## Coleção de Categoria de Produtos (*col\_product\_category*)

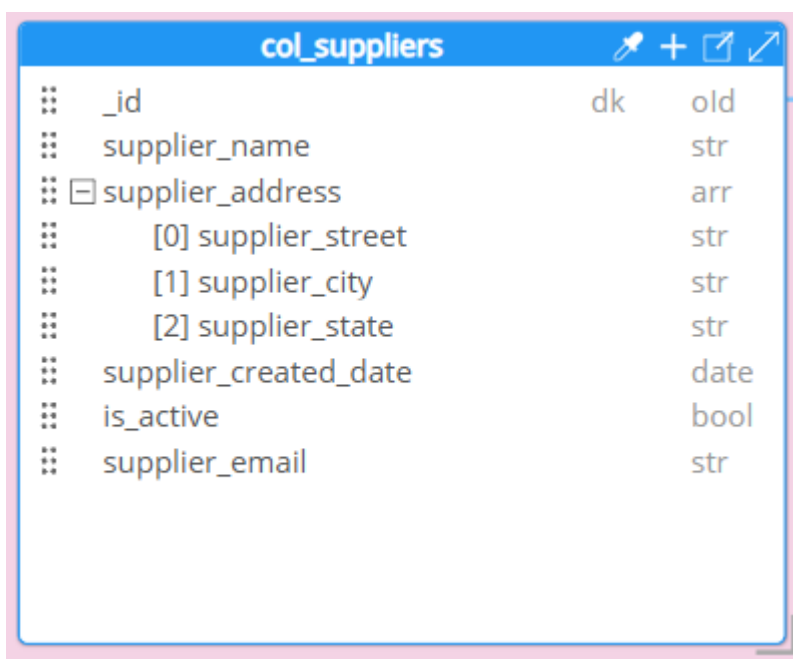


col_product_category		
::	_id	dk old
::	category_name	str
::	category_description	str

A coleção de Categoria de Produtos sumariza os agrupamentos que podem ser realizados dependendo dos produtos oferecidos pela empresa. Atua como coleção dimensão e pode ser combinada com a coleção pedidos através do id de categoria. Os nomes dos campos, seus respectivos formatos e obrigatoriedade são definidos como:

Campo	Formato	Obrigatoriedade
_id	Object ID	Y
Category_name	String	Y
Category_description	String	N

## Coleção de Fornecedores (*col\_suppliers*)

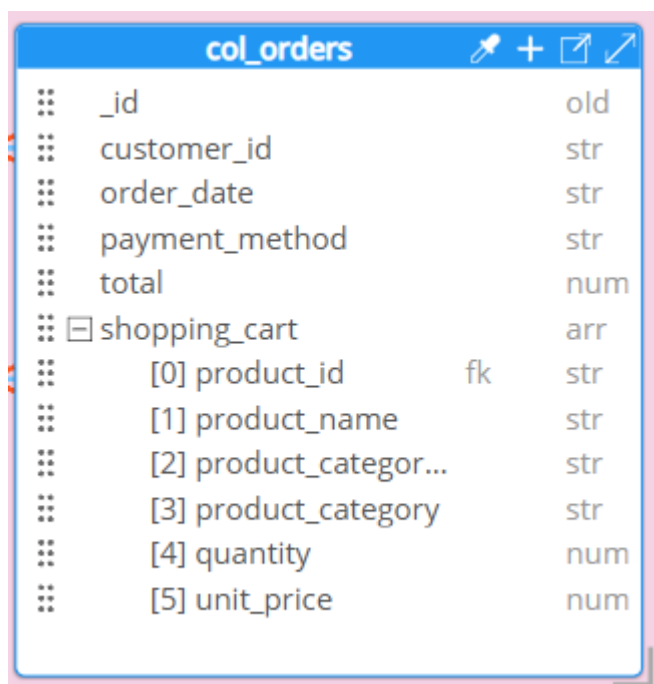


col_suppliers		
::	_id	dk old
::	supplier_name	str
::	supplier_address	arr
::	[0] supplier_street	str
::	[1] supplier_city	str
::	[2] supplier_state	str
::	supplier_created_date	date
::	is_active	bool
::	supplier_email	str

A coleção de Fornecedores é composta por identificadores do mesmo e uma coleção de endereço, possibilitando planejamentos estratégicos relacionados a logística, semelhante a coleção de endereço de clientes. Os nomes dos campos, seus respectivos formatos e obrigatoriedade são definidos como:

Campo	Formato	Obrigatoriedade
_id	Object ID	Y
Supplier_name	String	Y
Supplier_address	Array	Y
Supplier_address.supplier_street	String	Y
Supplier_address.supplier_city	String	Y
Supplier_address.supplier_state	String	Y
Supplier_created_date	Timestamp	Y
Is_active	Boolean	Y
Supplier_email	String	Y

### Coleção de Pedidos (*col\_orders*)



col_orders		
::	_id	old
::	customer_id	str
::	order_date	str
::	payment_method	str
::	total	num
::	shopping_cart	arr
::	[0] product_id	fk str
::	[1] product_name	str
::	[2] product_categor...	str
::	[3] product_category	str
::	[4] quantity	num
::	[5] unit_price	num

A coleção de Pedidos será composta por identificadores do cliente, a data do pedido, método de pagamento, o total do pedido e uma coleção contendo as

informações do carrinho detalhando produto a produto e suas quantidades/preço. Os nomes dos campos, seus respectivos formatos e obrigatoriedade são definidos como:

Campo	Formado	Obrigatoriedade
_id	Object ID	Y
Customer_id	String (FK)	Y
Order_date	Timestamp	Y
Payment_method	String	Y
Total	Number	Y
Shoping_cart	Array	Y
Product_id	String (FK)	Y
Product_name	String	Y
Product_category_id	String (FK)	N
Product_category	String	N
Quantity	Number	Y
Unit_price	Number	Y



## Plano de Escalabilidade

O banco será implementado no Atlas, serviço de database multi-cloud com distribuição compatível com as clouds públicas mais relevantes no mercado, Azure, AWS e Google Cloud.

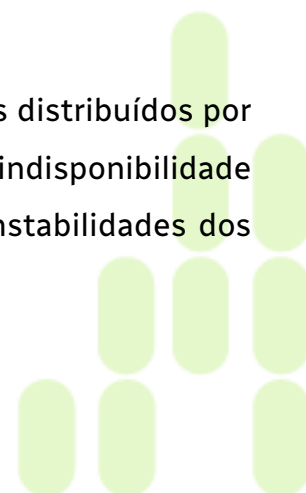
Com isso é possível aliar o desempenho do sistema com ferramentas de monitoramento, como Azure Monitor por exemplo, que coleta, analisa e pode até responder a dados de monitoramento, garantindo que a aplicação escale horizontalmente e possua quantidades de recursos razoáveis de acordo com a variação de demanda do mercado digital.

Além disso, será determinado um período de retenção dos dados de modo que tenhamos somente o volume necessário na aplicação. Após 5 anos por exemplo, é possível realizar um dump das collections em arquivos .json para o Azure Blob Storage em Cool ou Cold tier, que são estilos de armazenamento de baixo custo e que consideram que os arquivos não sejam acessados com tanta frequência:

- **Hot tier** - An online tier optimized for storing data that is accessed or modified frequently. The hot tier has the highest storage costs, but the lowest access costs.
- **Cool tier** - An online tier optimized for storing data that is infrequently accessed or modified. Data in the cool tier should be stored for a minimum of **30** days. The cool tier has lower storage costs and higher access costs compared to the hot tier.
- **Cold tier** - An online tier optimized for storing data that is rarely accessed or modified, but still requires fast retrieval. Data in the cold tier should be stored for a minimum of **90** days. The cold tier has lower storage costs and higher access costs compared to the cool tier.
- **Archive tier** - An offline tier optimized for storing data that is rarely accessed, and that has flexible latency requirements, on the order of hours. Data in the archive tier should be stored for a minimum of **180** days.

Fonte: [Microsoft](#)

Por parte do próprio MongoDB no Atlas, ele já possui replica sets distribuídos por availability zones por padrão, portanto dificilmente irá acontecer indisponibilidade do sistema ou até mesmo perda de informações por conta de instabilidades dos servidores.





Outra técnica que será implementada é a de sharding, método de distribuição de dados em servidores ou shards para escalonamento horizontal. Isso aumenta a capacidade do sistema para suportar cargas maiores de demanda ao aumentar o throughput de leitura/gravação.

Revisando nosso modelo, temos as seguintes collections:

- Col\_clients
- Col\_orders
- Col\_product\_category
- Col\_product
- Col\_suppliers

Nem todas as collections precisam de sharding pois o ideal é que essa abordagem ocorra em collections que possuam grandes volumes de dados e consultas frequentes. Com isso, vamos aplicar sharding nas seguintes collections:

- Col\_orders, pois tende a ter alta taxa de escrita com os pedidos diários
- Col\_products, pois a proposta é que a loja ofereça uma grande quantidade de produtos, de A até Z no Brasil.

Sobre a escolha das chaves responsáveis para criação dos shards, em Col\_orders será escolhido o order\_id e em Col\_products o product\_id, pois são identificadores únicos e assim evitam hotspots (grande de concentração de registros para uma única chave), melhorando a escalabilidade horizontal sem a necessidade de redistribuição constante.