



CQRS

O CQRS, ou **Command Query Responsibility Segregation**, foi muito divulgado por Greg Young e envolve separar os comandos que processam regras de negócio e realizam mutação de estado, das consultas que retornam apenas uma projeção sobre os dados armazenados

"Because the term **command** is widely used in other contexts I prefer to refer to them as modifiers, you also see the term mutators"

Martin Fowler



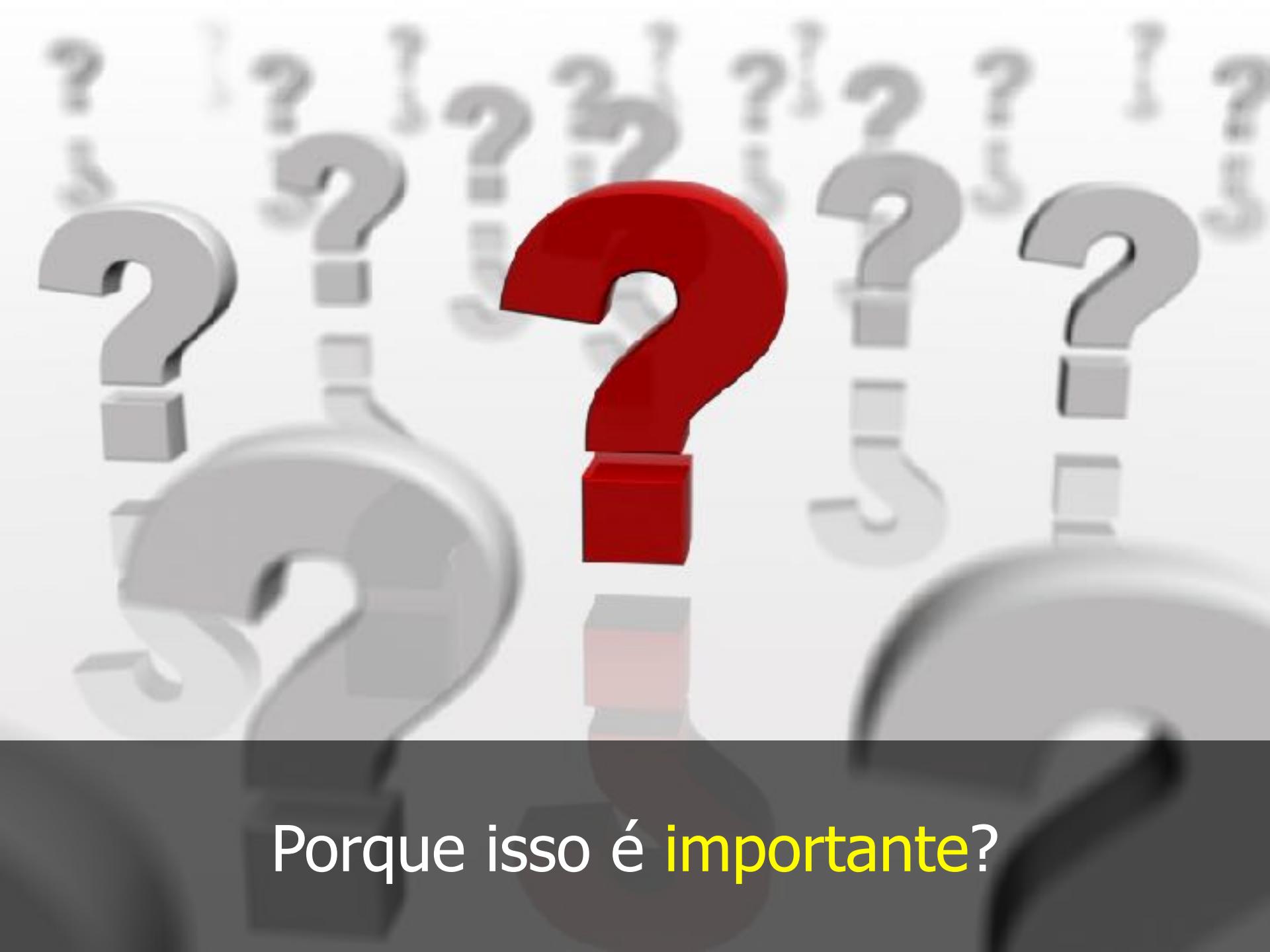
CQRS promove a **separação** entre o  
modelo de escrita e de leitura

# Comandos

- Implementam **regras de negócio**
- Fazem **mutação de dados**, ou seja, executam tarefas que envolvem inserir, atualizar ou deletar informações no banco de dados
- Eventualmente **podem ser colocados em uma fila e processados de forma assíncrona**
- Emitem **eventos** que desencadeiam processos de negócio complexos

# Consultas

- Não envolvem mutação de dados
- Retornam informações por meio de um DTO, ou Data Transfer Object
- Acessam os dados de uma forma simples e direta, ainda que seja a mesma base de dados utilizada pelos comandos
- Podem usar uma base de dados segregada, balanceada e eventualmente consistente



Porque isso é **importante**?



Domain-Driven Design ou qualquer outro  
tipo de orientação ao domínio



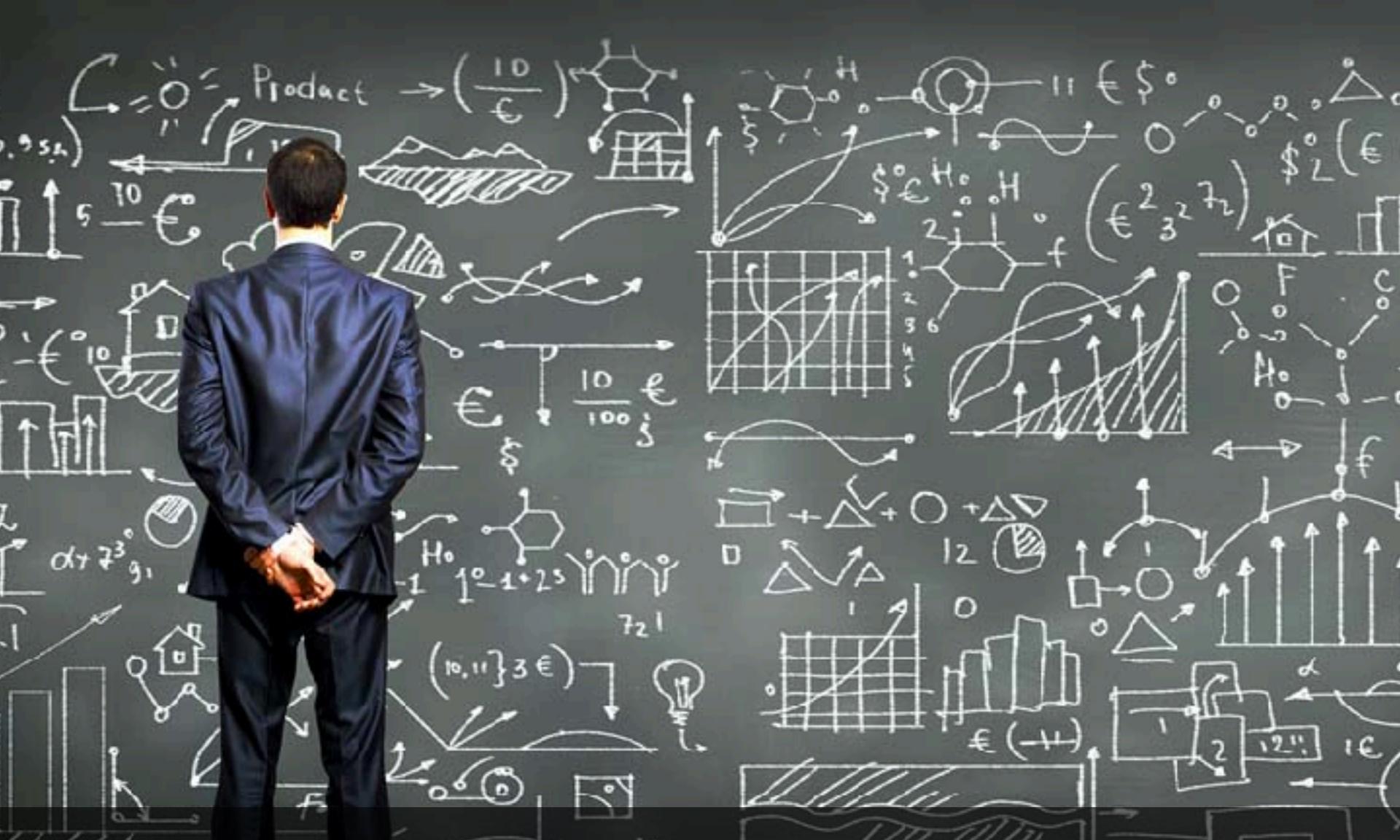
Normalmente, a persistência dos aggregates  
é feita por meio dos repositories



A granularidade do repository é por **aggregate**, que deve ser o menor possível



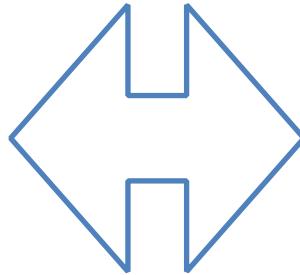
Todos os dados precisam ser obtidos por  
meio de um repositório?



Dependendo da consulta pode ser complexo  
obter dados de múltiplos aggregates

Lembre-se que a relação entre os aggregates é por identidade e dessa forma pode ser necessário obter diversos aggregates para consolidar as informações de retorno requeridas

```
{  
    "date": "2021-03-10",  
    "code": "2021000001",  
    "itens": [  
        {  
            "description": "Guitar",  
            "price": 1000,  
            "quantity": 1  
        },  
        {  
            "description": "Amplifier",  
            "price": 5000,  
            "quantity": 1  
        }  
    "total": 6000,  
    "freight": 300,  
    "customer": {  
        "name": "Rodrigo Branas",  
        "cpf": "11111111111",  
        "telephone": "99999999"  
    },  
    "taxes": [  
        {  
            "ipi": 120,  
            "icms": 200  
        }  
    ]  
}
```



DTO Assembly

Order

Item

Customer

Taxes



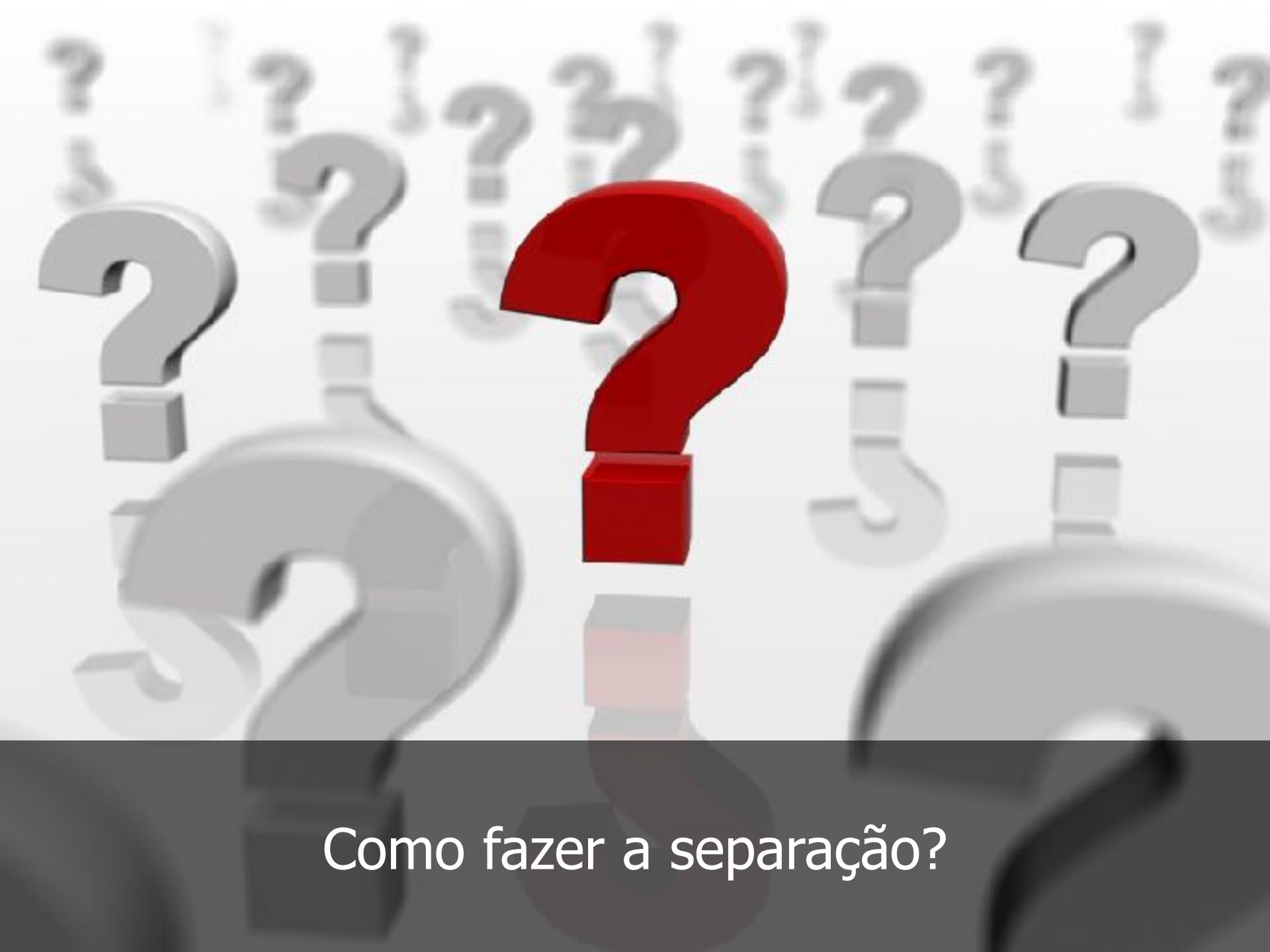
Com isso fazer com que objetos de domínio e  
repositórios **fiquem focados nas regras de negócio**



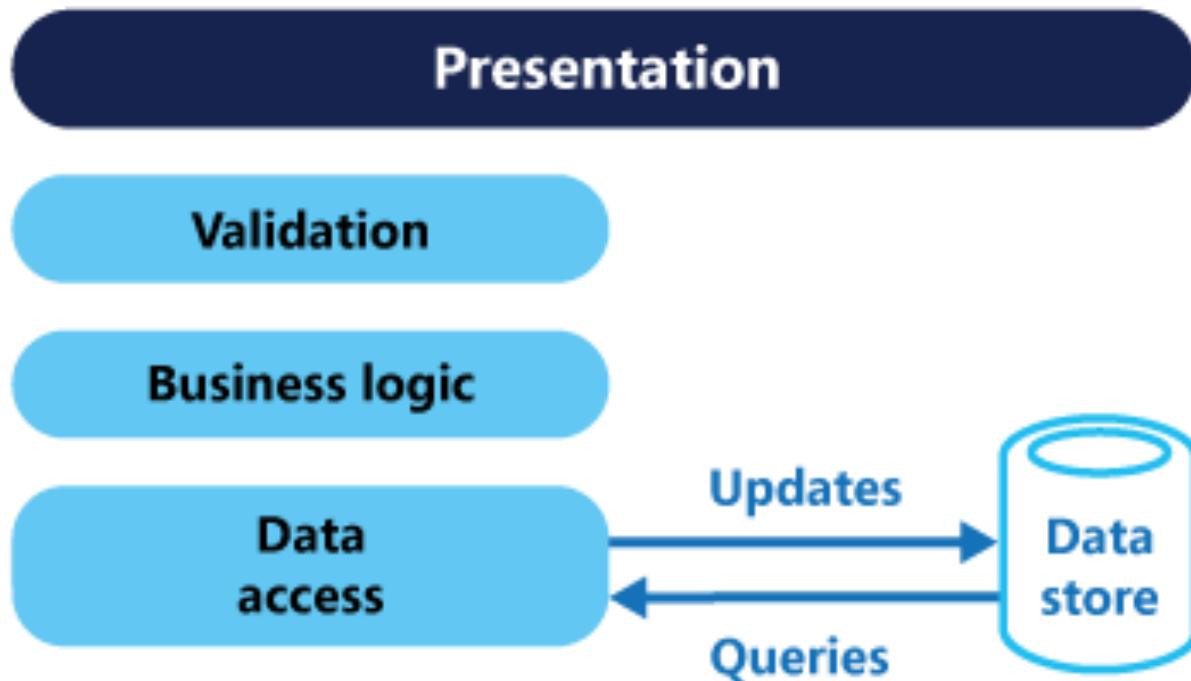
Microservices ou qualquer outro tipo de ambiente com **dados distribuídos**



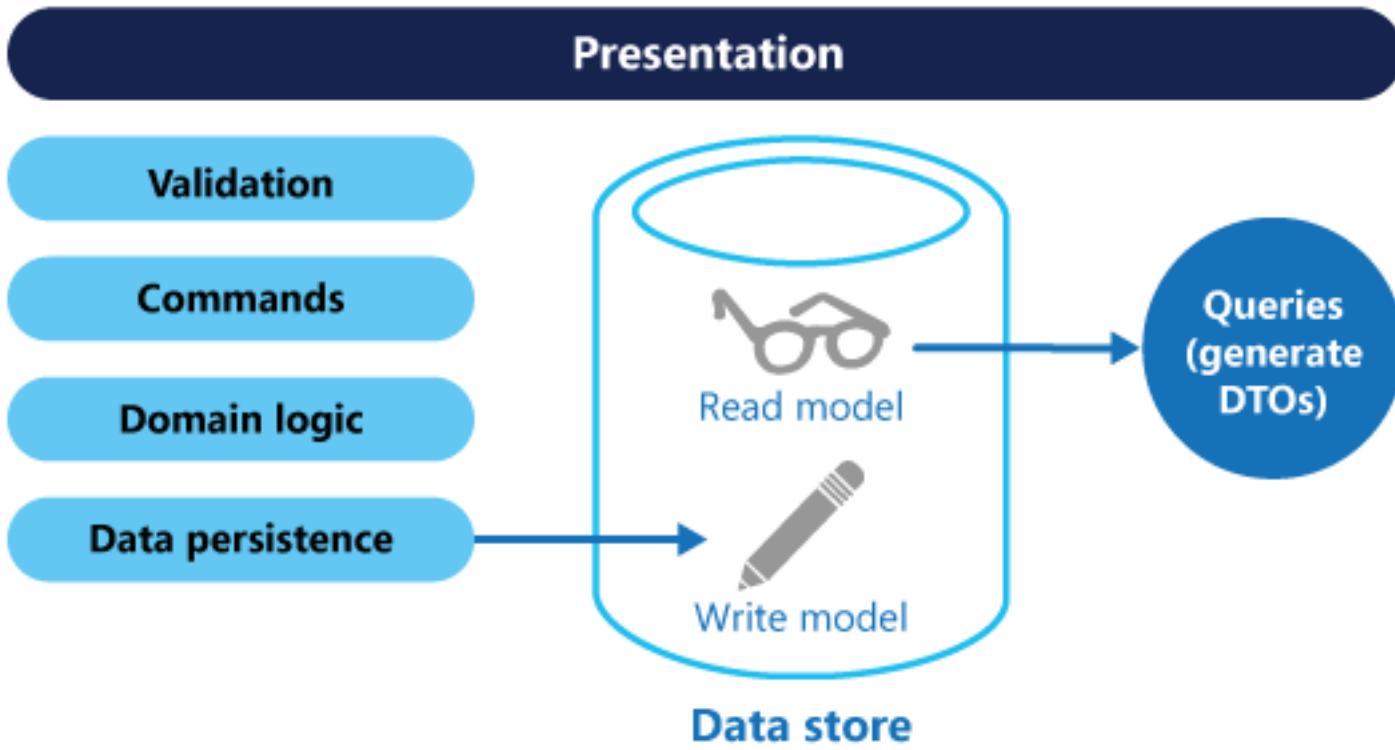
Facilitar a obtenção de dados, de forma escalável,  
reduzindo o acoplamento síncrono entre os serviços



Como fazer a separação?



<https://docs.microsoft.com/en-us/azure/architecture/patterns/cqrs>



<https://docs.microsoft.com/en-us/azure/architecture/patterns/cqrs>

## Presentation



<https://docs.microsoft.com/en-us/azure/architecture/patterns/cqrs>



Para utilizar CQRS precisamos obrigatoriamente  
**separar** os dados de escrita e de leitura?



CAUTION

Calma, não necessariamente...



O principal motivo é **performance**, o modelo de persistência nem sempre é otimizado para consulta



Qual é o papel da **normalização** em um banco de  
dados relacional?

# Normalização

- Reduzir a **duplicação** de dados entre diferentes tabelas, otimizando a ocupação de disco e também o risco de atualizar uma informação em uma tabela e esquecendo das outras
- Garantir a **consistência** nas operações realizadas sobre os dados
- Permite a **combinação** criando projeções especificadas dependendo das necessidades



A **normalização** tem mais relação com a escrita  
ou com a leitura dos dados?

Não é recomendado renderizar relatórios ou estatísticas complexas a partir do banco de dados utilizado para a escrita, principalmente se ele for relacional



O banco de dados de escrita precisa ser relacional  
e o de leitura precisa ser **NoSQL**?

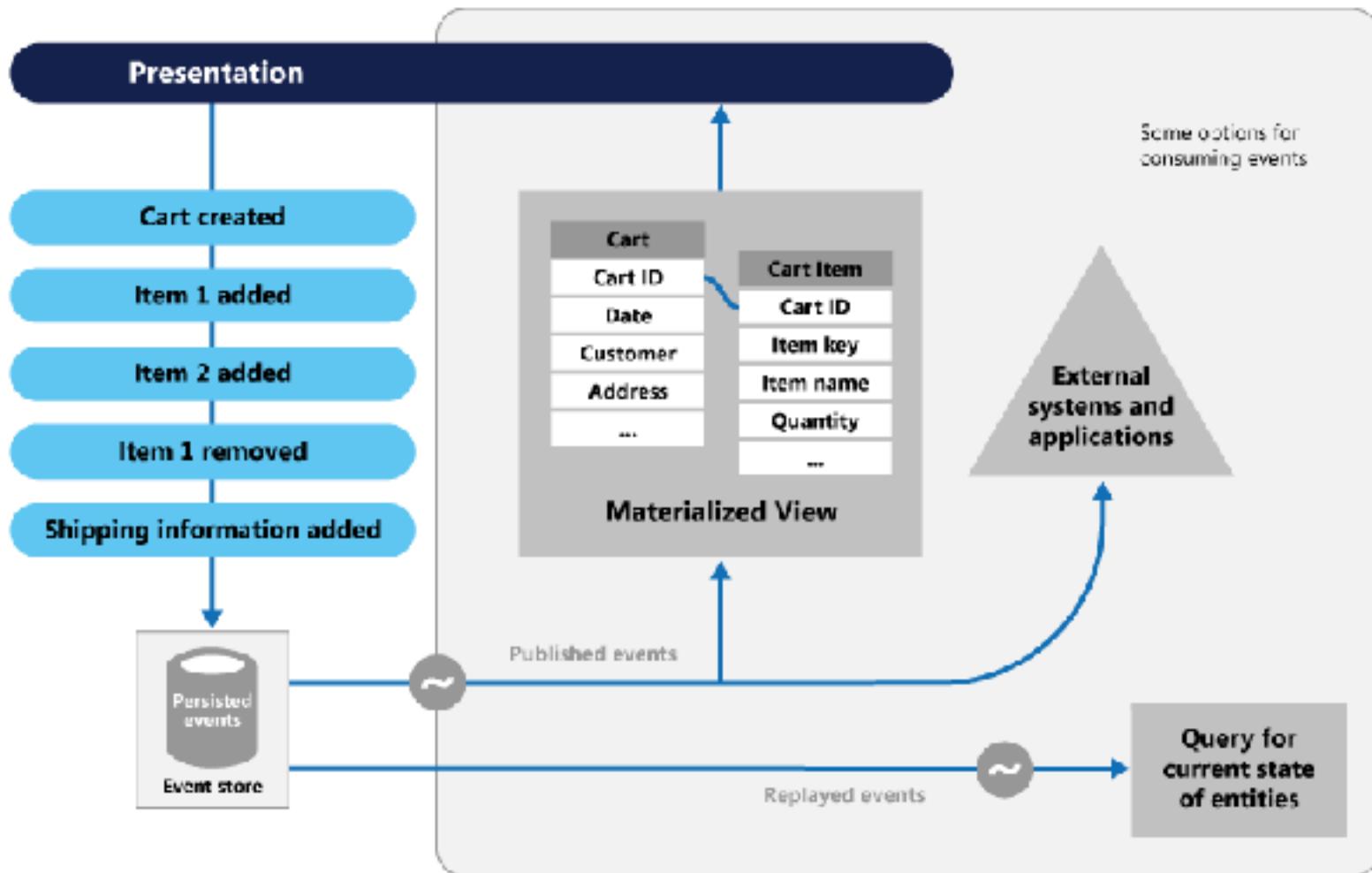


Se o objetivo é armazenar **múltiplas projeções**  
**não estruturadas**, talvez seja melhor usar NoSQL



Como manter o banco de leitura **sincronizado**?

É possível atualizar ao longo da transação de escrita de forma síncrona ou mesmo criar um mecanismo mais resiliente e assíncrono, publicando **eventos** e **consumindo em uma fila**



<https://docs.microsoft.com/en-us/azure/architecture/patterns/event-sourcing>