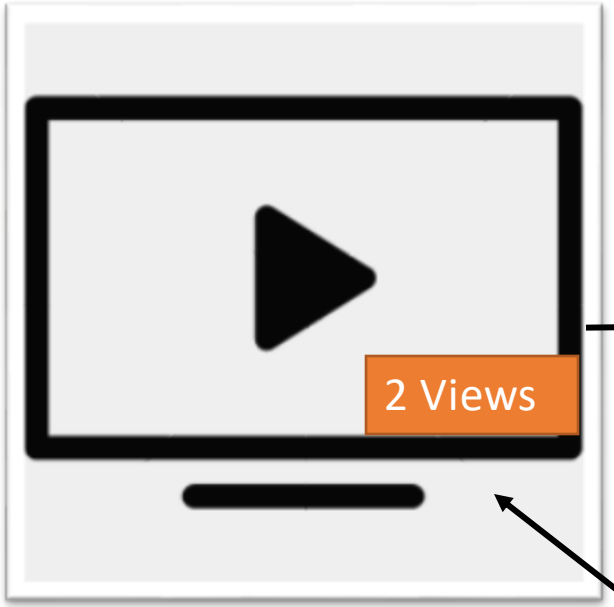
CQRS

7COMm
Serviços e Soluções em TI

Contexto e problema

- Nas arquiteturas tradicionais, utiliza-se o mesmo modelo de dados para escrita e consulta
- É possível existir formas diferentes no retorno de consultas, diferindo do modelo original de dados
- No lado da gravação, o modelo pode implementar uma validação complexa e lógica de negócios. Como resultado, você pode terminar com um modelo excessivamente complexo que faz coisas em excesso.

Contexto e problema



video_id	user_id	timestamp
1	100	1646874880
1	null	1646874881

```
SELECT Count(*) FROM VIDEO_ACCESS
WHERE VIDEO_ID = 1
```

Contexto e problema

video_id	user_id	timestamp
1	100	1646874880
1	null	1646874881

VIDEO_ACCESS

Contexto e problema

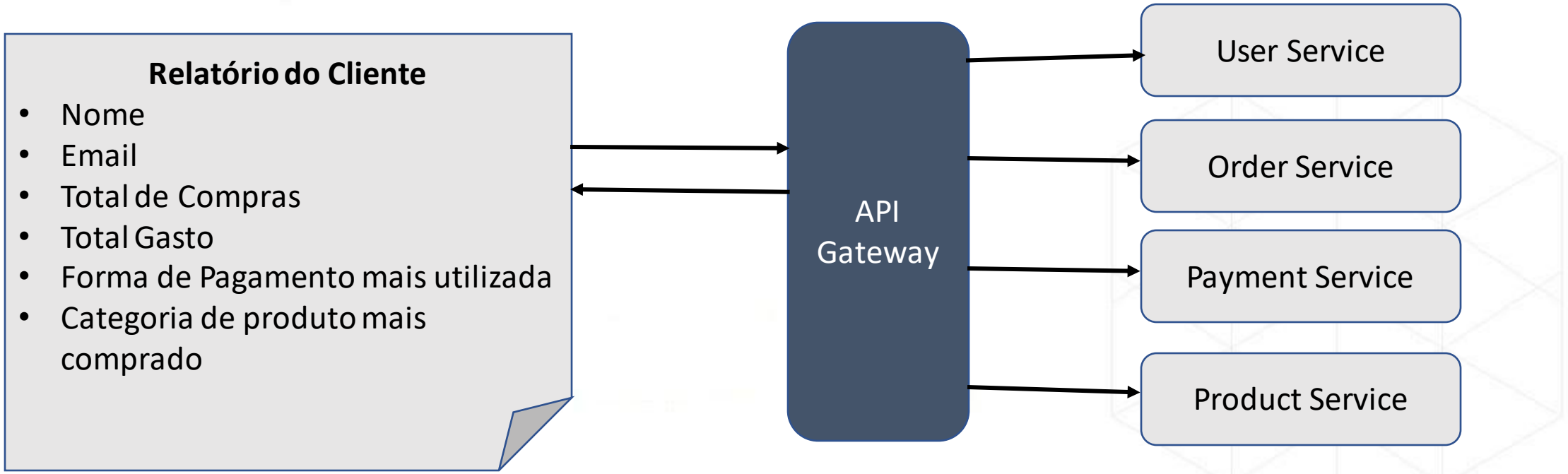
video_id	user_id	timestamp
1	100	1646874880
1	null	1646874881
1	null	1646874890
1	null	1646874881
1	null	1646874881
1	null	1646874881
1	null	1646874881
1	null	1646874881

VIDEO_ACCESS

Contexto e problema

- A escrita na tabela de visualizações concorre diretamente com a leitura dos dados agregados
- Pode-se observar problemas com lock da tabela e da leitura/escrita dos dados
- Acaba tornando-se uma funcionalidade ineficiente

Contexto e problema



Contexto e problema

- Deve ser feito Queries em cada serviço
- O trabalho de *merge* e *join* das queries deve ficar a cargo do API Gateway
- Caso tenha paginação e filtros, o problema se torna bem mais complicado

Contexto e problema

- Como garantir que operações de leitura não impactem as operações de escrita?
- Como tornar mais eficientes as buscas através de vários serviços?
- Como tornar o resultado de uma consulta mais aderente com o objetivo da minha tarefa?

CQS (Command Query Separation)

- Separação das tarefas de leitura e gravações em modelos diferentes, usando **comandos** para atualizar dados e **queries** para ler dados.
- Comandos **alteram** o estado dos objetos e têm efeito colateral
- Comandos **não retornam nada**
- Consultas são somente leitura
- Consultas em hipótese alguma alteram o estado de um objeto
- https://martinfowler.com/bliki/CommandQuerySeparation.html
- Proposto por Bertrand Meyer, em 1988, no livro Object Oriented Software Construction

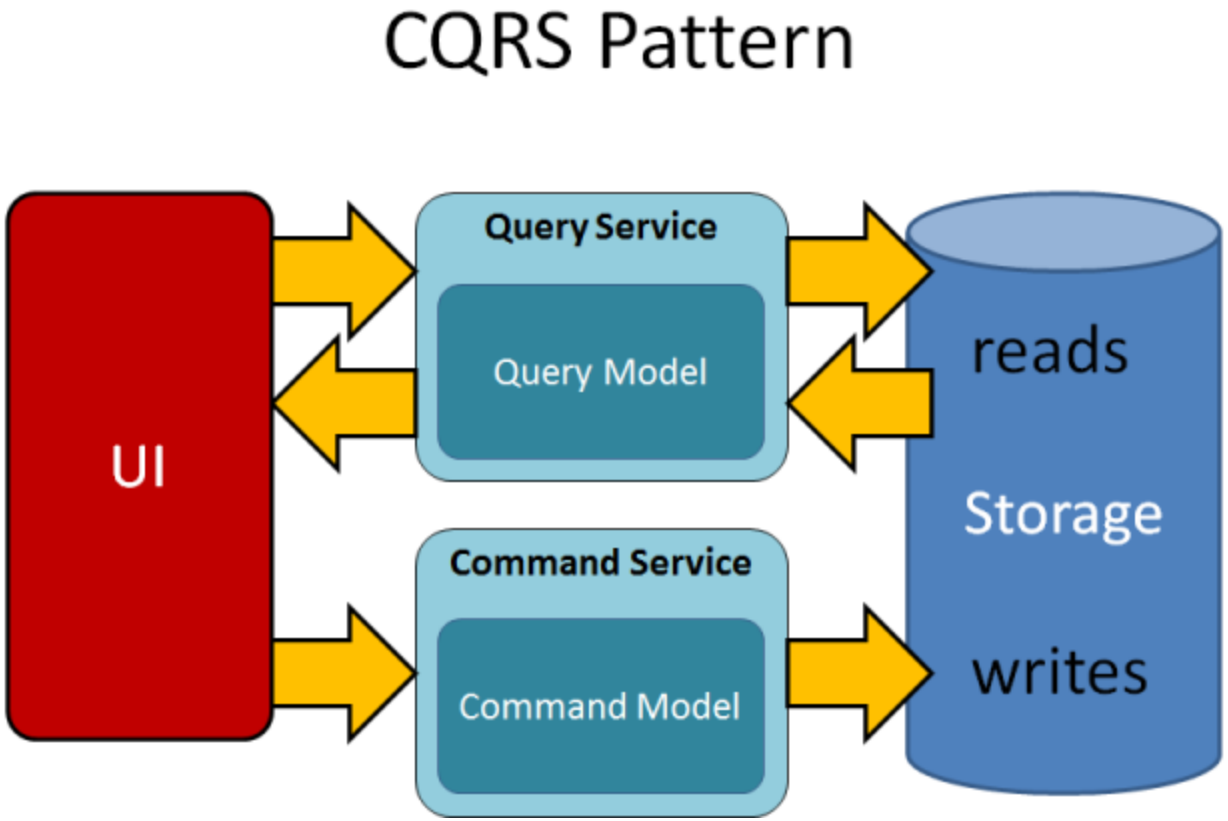
CQS

```
public class CustomerService
{
    // Command
    public void Process(string name, string address)
    {
        Address addr = CreateAddress(address);
        Customer customer = CreateCustomer(name, addr);
        SaveCustomer(customer);
    }

    // Query
    private Address CreateAddress(string address)
    {
        return new Address(address);
    }

    // Query
    private Customer CreateCustomer(string name, Address address)
    {
        return new Customer(name, address);
    }

    // Command
    private void SaveCustomer(Customer customer)
    {
        var repository = new Repository();
        repository.Save(customer);
    }
}
```

CQRS

- **Command Query Responsibility Segregation**
- Padrão arquitetural
- Não atua a nível de código, mas a nível de aplicação
- Pode maximizar o desempenho da aplicação, escalabilidade e segurança.
- Impede que os comandos de atualização causem conflitos de merge no nível de domínio.
- http://microservices.io/patterns/data/cqrs.html

Esquema CQRS



Benefícios CQRS

- Dimensionamento independente
  - Bancos de leitura e gravação podem ser otimizados com recursos compatíveis com sua utilização
- Esquemas de dados otimizados
- Segurança
  - É mais fácil garantir que apenas as entidades do direito de domínio estejam executando gravações nos dados.
- Consultas mais simples

Benefícios CQRS

- Divisão de problemas
  - Modelos mais flexíveis
  - A maior parte da lógica de negócios complexa vai para o modelo de gravação
  - O modelo de leitura pode ser consideravelmente mais simples.
- Ao utilizar views materializadas, evita-se o uso exaustivo de joins
- Permite queries em uma aplicação baseada em Event Sourcing

Problemas de implementação CQRS

- Complexidade de desenvolvimento
  - Apesar da ideia simples, pode representar uma mudança drástica no formato do desenvolvimento
  - Atua diretamente com a complexidade de Event Sourcing.
- Mensagens
  - É comum a utilização de mensageria comandos e publicação eventos.
  - Deve-se tratar as falhas ou as mensagens duplicadas.
- Consistência eventual
  - Dados de leitura constantemente obsoletos
  - Há uma complexidade a mais para a sincronização dos modelos

Quando usar CQRS

- Domínios colaborativos em que muitos usuários acessam os mesmos dados em paralelo.
- O modelo de gravação tem uma pilha completa de processamento de comandos com lógica de negócios, validação de entrada e validação de negócios.
- Cenários em que o desempenho de leituras de dados deve ser ajustado separadamente do desempenho de gravações de dados, especialmente quando o número de leituras é muito maior do que o número de gravações.

Quando usar CQRS

- Cenários onde uma equipe de desenvolvedores pode se concentrar no modelo de domínio complexo de gravação e outra equipe pode se concentrar no modelo de leitura e nas interfaces de usuário.
- Cenários onde o sistema deve evoluir ao longo do tempo e pode conter várias versões do modelo, ou onde as regras de negócio mudam regularmente.
- Integração com outros sistemas, especialmente em combinação com *event sourcing*, onde a falha temporal de um serviço não deve afetar a disponibilidade dos outros.

Quando não usar CQRS

- O domínio ou as regras de negócio são simples.
- Quando uma interface de usuário simples no estilo CRUD e operações de acesso a dados são suficientes.

View CQRS

- Serviço responsável por realizar a consulta propriamente dita
- Consiste em uma ou mais operações de leitura
- Fornece uma API para executar essas operações em sua base de dados
- Sua base de dados é alimentada através de eventos publicados, de um ou mais serviços

View CQRS

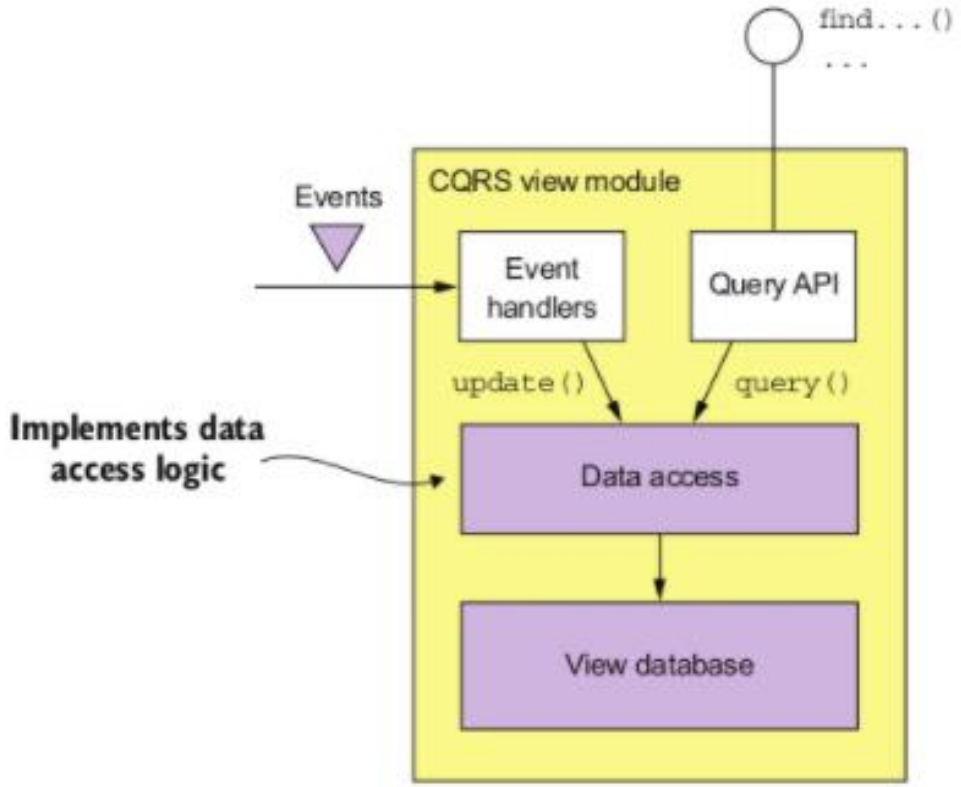


Figure 7.10 The design of a CQRS view module. Event handlers update the view database, which is queried by the Query API module.

View CQRS

- O módulo Data Access implementa a lógica do banco de dados
- *Event Handlers* atualizam o banco
- Query API, a consulta propriamente dita

View CQRS

- O módulo Data Access implementa a lógica do banco de dados
- *Event Handlers* atualizam o banco
- Query API, a consulta propriamente dita
- Nesse ponto, passamos por decisões importantes para nossa implementação
  - A escolha do banco de dados e dos respectivos schemas

SQL vs NoSQL

- As bases SQL dominaram o desenvolvimento até pouco tempo atrás
- No entanto, conforme a web crescia, vários requisitos acabaram não sendo totalmente satisfeitos com a utilização dessa tecnologia
- Criou-se então, as bases NoSQL (Not only SQL)
- Bases NoSQL normalmente não são tão versáteis para implementação de queries, porém apresentam schemas flexíveis e melhor performance e escalabilidade

SQL vs NoSQL

- A escolha da base em nossa arquitetura CQRS cabe a necessidade que temos em mãos
- As vezes podemos nos beneficiar da versatilidade e performance de um modelo NoSQL, as vezes podemos nos beneficiar da versatilidade em queries do modelo SQL
- Hoje em dia, bases SQL fornecem suporte a features NoSQL, como tipos de dados geoespaciais e suporte a objetos json

SQL vs NoSQL

Necessidade	Uso
Baseado em chave primária e dados de pesquisa dentro de objetos json	Base de dados de documentos, como MongoDB, ou um banco *chave-valor,* como Redis
Baseado em consultas e dados de pesquisa dentro de objetos json	Uma base de documentos, como MongoDB
Queries de texto	Uma engine de busca, como Elastic Search
Consultas em grafos	Uma base de dados de grafos, como Neo4j
Relatórios tradicionais baseados em tabelas	Banco relacional

Axon Framework

- Framework utilizado para auxiliar no desenvolvimento de aplicações com CQRS e Event Sourcing
- Tenta prover uma maneira unificada e produtiva para evoluir aplicações o modelo orientado a eventos
- Possui uma versão open source
- https://developer.axoniq.io/

Para Saber mais...

- https://danylomeister.blog/2020/06/25/cqs-cqrs-event-sourcing-whats-the-difference/
- https://developer.axoniq.io/
- http://microservices.io/patterns/data/cqrs.html
- https://spring.io/guides/gs/gateway/

OBRIGADO!

Centro
Rua Formosa, 367 - 29º andar Centro, São Paulo - SP, 01049-000

Alphaville
Avenida Ipanema, 165 - Conj. 113/114 Alphaville, São Paulo - SP,06472-002

+55 (11) 3358-7700
contact@7comm.com.br

7COMm
Serviços e Soluções em TI