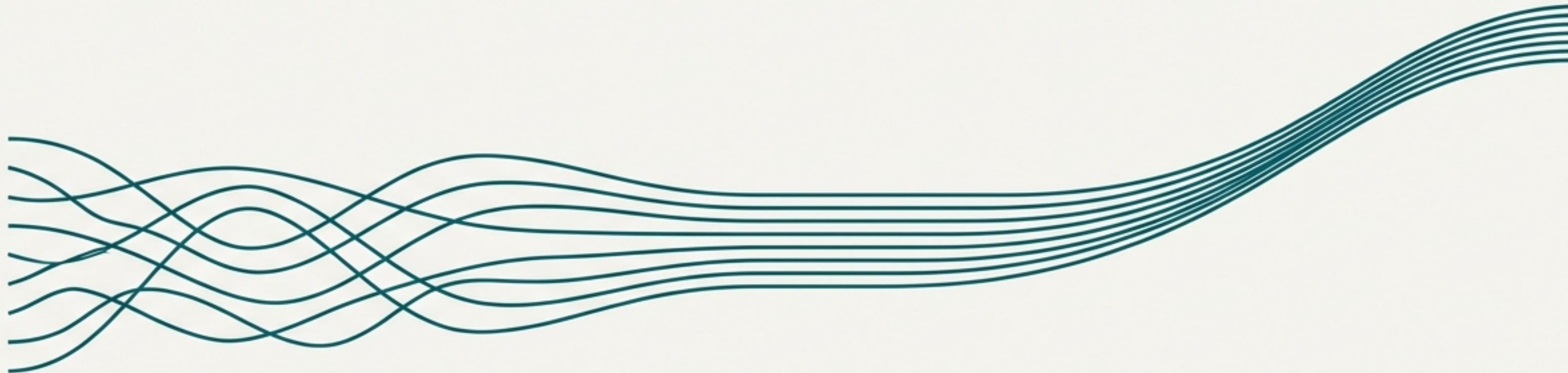


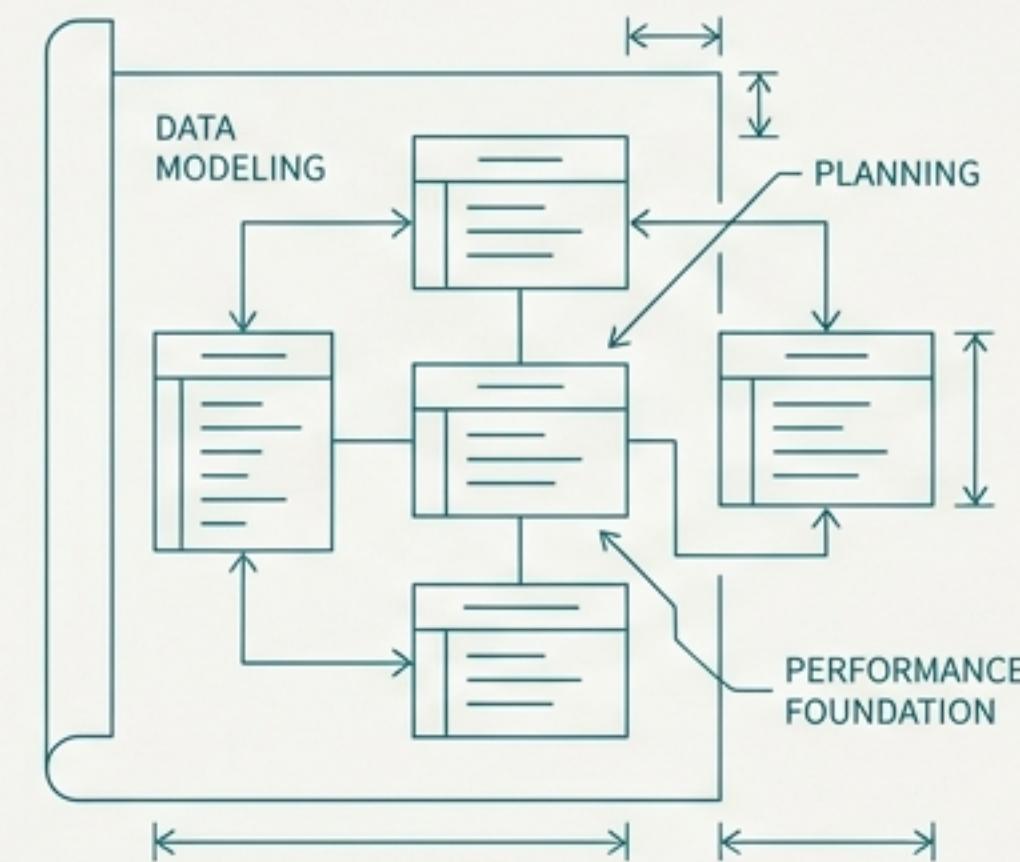
Modelagem de Performance em CAP

Um Guia Prático para Desenvolvedores Modernos



Performance é um Atributo de Design, Não um Acidente

Decisões de modelagem ruins no início do projeto se transformam em gargalos de performance, código complexo e dívida técnica no futuro. Nosso objetivo não é apenas construir aplicações que funcionam, mas sim aplicações que são performáticas, escaláveis e fáceis de manter. Este guia é o seu blueprint para alcançar a excelência em performance desde a fundação do seu modelo de dados.



Nossa Jornada: Dos Erros Comuns às Melhores Práticas



I. Estruturas Fundamentais

- Evitando a ineficiência do `UNION` com Polimorfismo.
- O uso correto de Composições vs. Associações.



II. Otimização de Consultas

- Dominando `JOINS`, `SORTs` e `FILTERs`.
- A abordagem correta para Campos Calculados.



III. Modernização de Padrões

- Refatorando hábitos de sistemas legados para a era CAP.

O Anti-Padrão `UNION`: O Custo Oculto da Conveniência

O uso de `UNION` em views deve ser evitado, especialmente se operações como `SORT` ou `FILTER` são aplicadas sobre o resultado. Embora possa parecer uma solução rápida, ele introduz complexidade e degrada a performance.

Casos de Uso Comuns (e Problemáticos):

1. Implementação de Polimorfismo.
2. Portabilidade de aplicações legadas que já utilizam `UNION`.

AVISO

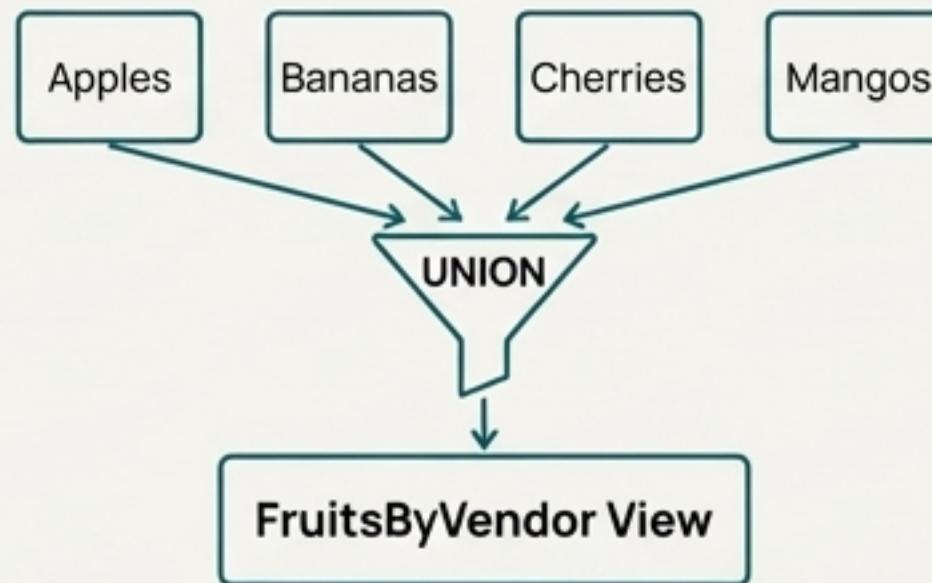
UNIONS em views carregam uma penalidade de performance e resultam em uma modelagem complexa.

Regra de Ouro

Regra de Ouro Ao iniciar um novo modelo, você nunca deveria precisar usar `UNION`.

'UNION' vs. Polimorfismo: Uma Batalha pela Eficiência

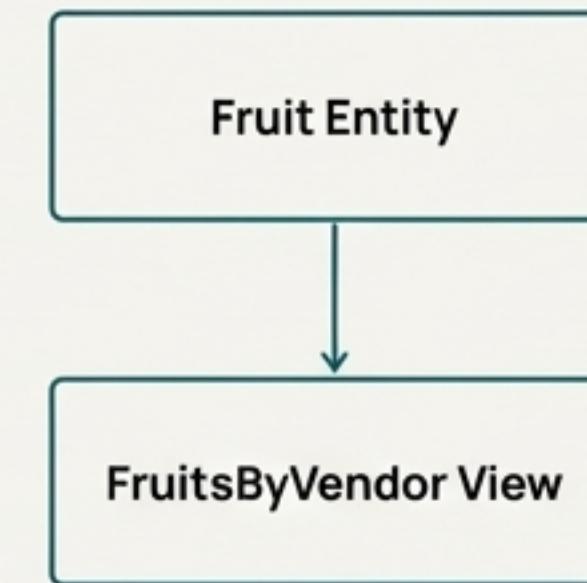
O Padrão Problemático: Múltiplas Entidades com 'UNION'



```
// Múltiplas entidades separadas
entity Apples : cuid, managed { ... }
entity Bananas : cuid, managed { ... }
entity Cherries : cuid, managed { ... }
entity Mangos : cuid, managed { ... }

// View complexa com UNION
view FruitsByVendor as
    select from Apples UNION
    select from Bananas UNION
    select from Cherries UNION
    select from Mangos
    {ID, description, vendor}
    where vendor.description = 'TopFruitCompany';
```

A Solução Moderna: Entidade Única com Polimorfismo



```
// Entidade única e de-normalizada
entity Fruit : apple, banana, cherry, mango, cuid, managed {
    type      : String enum { apple; banana; cherry; mango };
    description : String;
    vendor    : Association to one Vendor;
}

// View drasticamente simplificada
view FruitsByVendor as
    select from Fruit
    {ID, description, vendor}
    where vendor.description = 'TopFruitCompany';
```

O Resultado: Menos associações para gerenciar, zero 'UNION's em suas queries e views mais simples e performáticas.

Composições vs. Associações: Definindo Relacionamentos Sólidos

Use Composições Quando...

- O ciclo de vida do pai e do filho é compartilhado.
- Existe uma hierarquia clara de "pai-filho".
- A entidade filha nunca é exposta por conta própria.
- Você deseja manter as entidades transacionalmente juntas.

Use Associações Quando...

- As entidades têm ciclos de vida independentes.
- Não há uma hierarquia clara de "pai-filho".
- As entidades filhas são expostas de forma independente.
- Os relacionamentos podem mudar ao longo do tempo.



Regra de Ouro

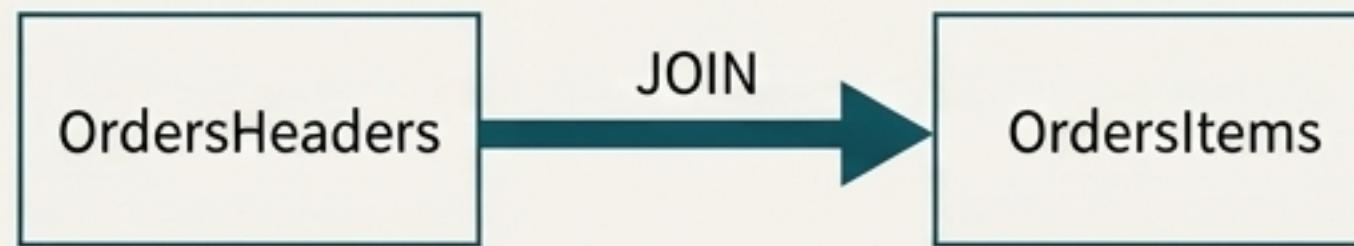
Seu braço é uma **composição** do seu corpo. Seu smartphone é uma **associação** com você, pois amanhã ele pode pertencer a outra pessoa.

CUIDADO Documentos grandes com milhares de filhos em composições são copiados *inteiramente* para o estado de rascunho (draft). Em tais casos, desacople o documento usando associações para evitar gargalos.

O Dilema do `JOIN`: A Projeção Dinâmica é a Resposta

Bad: Junção Estática com `JOIN`

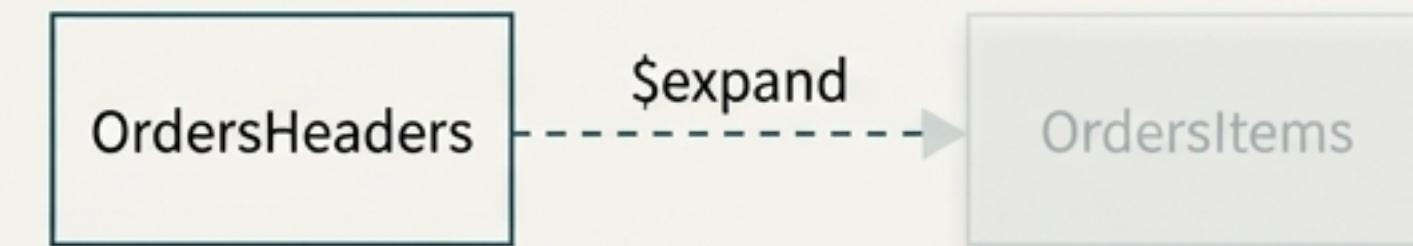
O `JOIN` é executado **sempre**, mesmo que os dados dos itens não sejam necessários.



```
view OrdersItemsViewJoin as select
    OrdersHeaders.ID      as Header_ID,
    OrdersHeaders.OrderNo  as OrderNo,
    OrdersItems.ID         as Item_ID,
    OrdersItems.product    as product,
    ...
from OrdersHeaders JOIN OrdersItems on OrdersHeaders.ID =
    OrdersItems.Header.ID;
```

Good: Projeção Dinâmica com Associação

O `JOIN` só é executado sob demanda via OData.



```
entity OrderItemsViewAssoc as projection on OrdersHeaders;
GET .../OrderItemsViewAssoc?$expand=Items&$select=OrderNo,Items
```

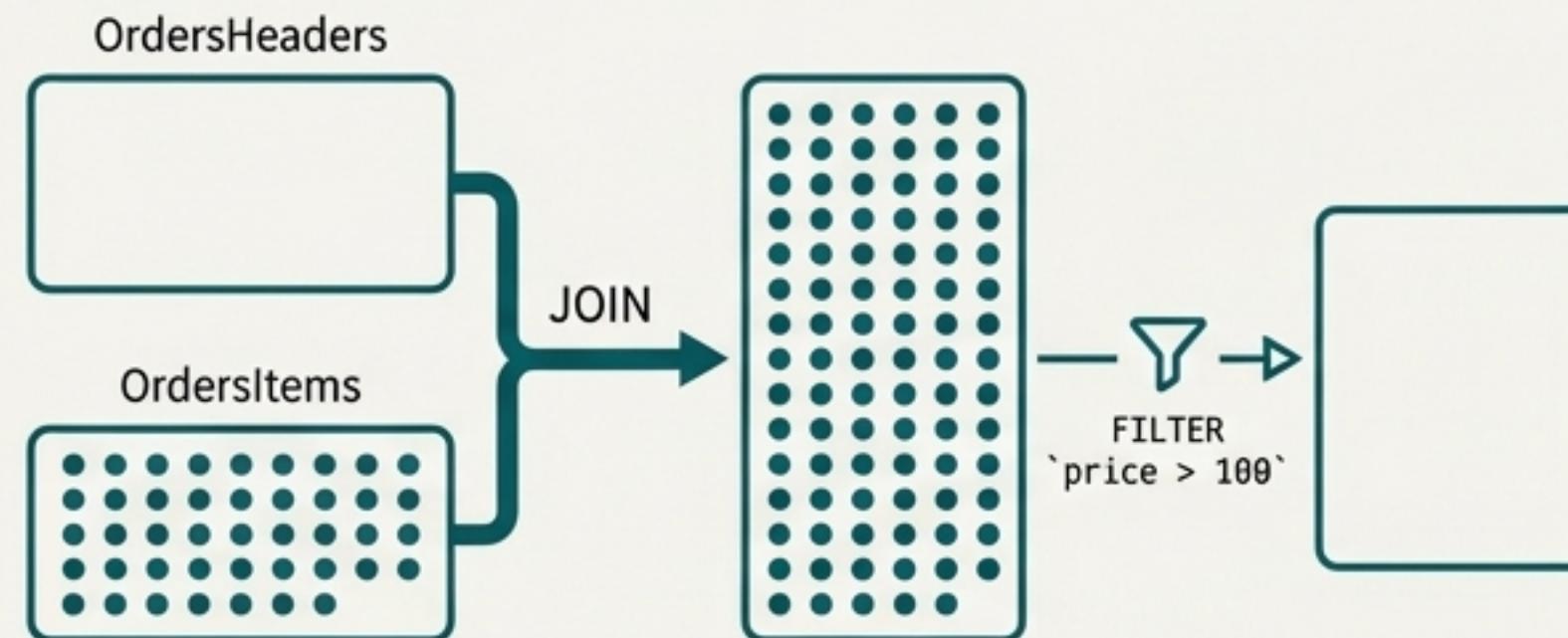
Com a projeção dinâmica, você tem controle total. O `JOIN` só ocorre quando explicitamente solicitado com `'\$expand'`, buscando apenas o que é necessário e quando é necessário.

Otimizando `SORT` e `FILTER`: A Ordem das Operações Importa

A chave para a performance é reduzir o volume de dados na tabela mais granular *antes de realizar a junção (`JOIN`) com outras tabelas.

Bad: Filtrar Depois do `JOIN`

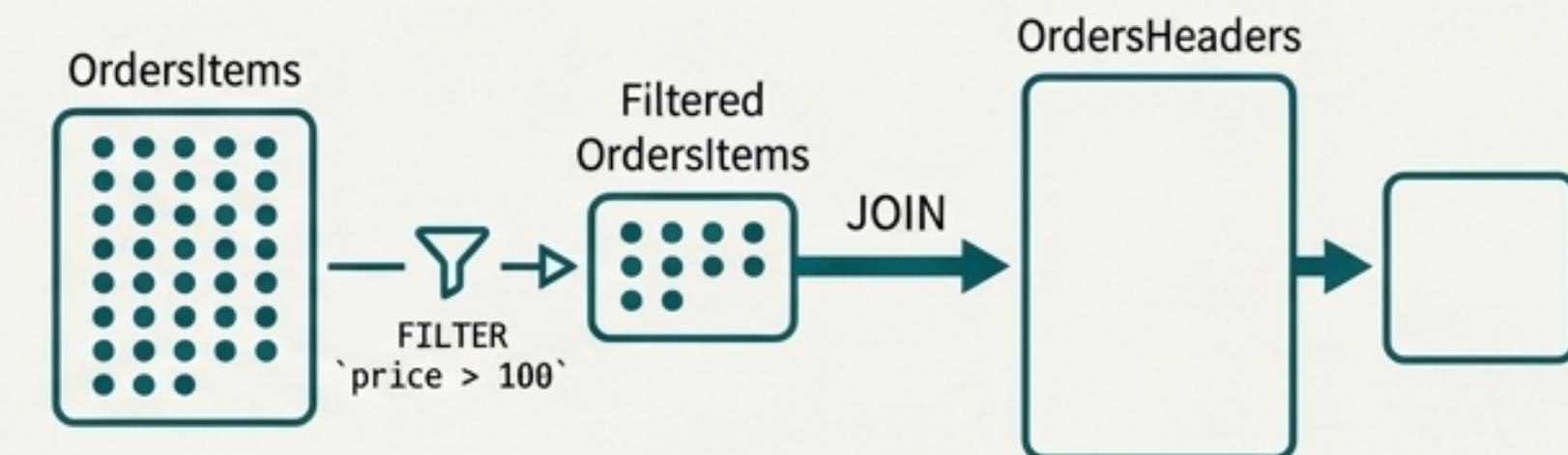
O banco de dados materializa a junção completa de `OrdersHeaders` e `OrdersItems` em memória *antes de aplicar o filtro `price > 100`*.



```
view FilteredOrdersJoin as select ...
  from OrdersHeaders JOIN OrdersItems on ...
  where price > 100;
```

Good: Filtrar Antes do `JOIN`

A subconsulta filtra `OrdersItems` primeiro, criando um conjunto de resultados muito menor, e só então associa de volta a `OrdersHeaders`.



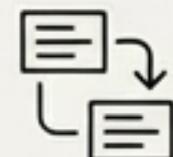
```
view FilteredOrdersAssoc as select *, Header.OrderNo, ...
  from (
    select from OrdersItems {*}
      where OrdersItems.price > 100
  );
```

Filtrar na tabela certa primeiro evita que a junção completa seja materializada, resultando em um ganho massivo de performance. O mesmo princípio se aplica ao `SORT`.

Campos Calculados: A Armadilha da Performance em Tempo Real

Operações de banco de dados em campos calculados não podem aproveitar índices, resultando em ***full table scans*** e impactando a performance significativamente.

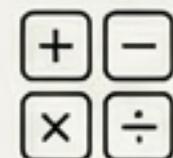
Culpados Comuns



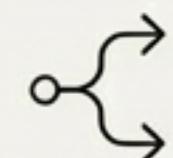
Concatenação de Strings: `PrintedName = concat.FirstName, LastName`



Formatação: IBAN com espaços - DE99 6611...



Álgebra: `FinalPrice = Rebate * ListPrice`



Declarações `CASE`: `case when quantity > 100 then 'Medium' ... end`

Evite usar campos calculados em tempo real em cláusulas `WHERE`, condições de `JOIN` ou filtros de associação.

A Hierarquia da Calculação: Da UI à Pré-computação

Existe um lugar certo e uma hora certa para cada cálculo.
Escolha a opção mais performática sempre que possível.

Hierarquia de Preferência



Comparação Visual

Bad (`on read`)

```
// service.cds
entity OrdersItemsView as projection on OrdersItems {
  *,
  case when quantity > 100 then 'Medium' ... end as category
};
```

Good (`on write`)

```
// schema.cds
extend my.OrdersItems with {
  category: String = case ... end stored;
}
```

Regra de Ouro: Calcule uma vez na escrita, leia muitas vezes. Evite recalcular a cada leitura.

Quebrando os Hábitos do Legado: Remodelar para Vencer

Muitos anti-padrões de performance são herdados de sistemas legados, onde foram criados para contornar limitações de hardware ou de banco de dados que não existem mais. Não transfira esses padrões para suas novas aplicações CAP. O esforço de refatorar o modelo de dados é recompensado com performance, simplicidade e código mais limpo.

Legado → Remodelar → Moderno → Melhor Performance

Do Legado ao Moderno: Guia de Refatoração (Parte 1)

Padrão Legado

Booleans em String ('X' para `true`, ' ' para `false`).



Solução Moderna

Converta para o tipo `Boolean` nativo do banco de dados. Isso simplifica a lógica e evita futuras declarações `CASE`.

Padrão Legado

Decimais emulados (inteiros com informações de formatação).



Solução Moderna

Converta para o tipo `Decimal` nativo. A conversão elimina a complexidade da aplicação.

Padrão Legado

Múltiplas colunas de atributos (`Endereco01`, `Endereco02`, ...).



Solução Moderna

Use composições de um tipo estruturado. Isso simplifica queries e a lógica de negócio.

Do Legado ao Moderno: Guia de Refatoração (Parte 2)

Padrão Legado

Views de abstração desnecessárias (`I_VIEW`, `C_VIEW`).



Solução Moderna

Use a separação de camadas nativa do CAP (`db` e `srv`).
Projete entidades para persistência otimizada e serviços para consumo otimizado.

Padrão Legado

Estruturas de dados monolíticas e complexas.



Solução Moderna

Mantenha as entidades simples e com um único propósito. Use `Actions` e `Functions` para manipulações de dados mais complexas.

Padrão Legado

`UNION` e `CASE` statements.



Solução Moderna

Remodele seu domínio de dados conforme descrito nas seções de Polimorfismo e Campos Calculados.

Seus Princípios para uma Modelagem de Alta Performance



Prefira Polimorfismo a `UNION`'s. Modele entidades semanticamente relacionadas em uma única estrutura.



Reduza o volume de dados *antes* de juntar, ordenar ou filtrar. A ordem das operações é crucial.



Calcule na escrita, não na leitura. Armazene resultados pré-calculados sempre que possível.



Use os tipos de dados nativos do banco de dados. Evite emulações que adicionam complexidade e degradam a performance.



Modele para simplicidade e um propósito único. Entidades e serviços enxutos são mais fáceis de manter e otimizar.

Modele com Intenção. Construa para o Futuro.

A performance de amanhã é definida pelas decisões
de modelagem que você toma hoje.

