

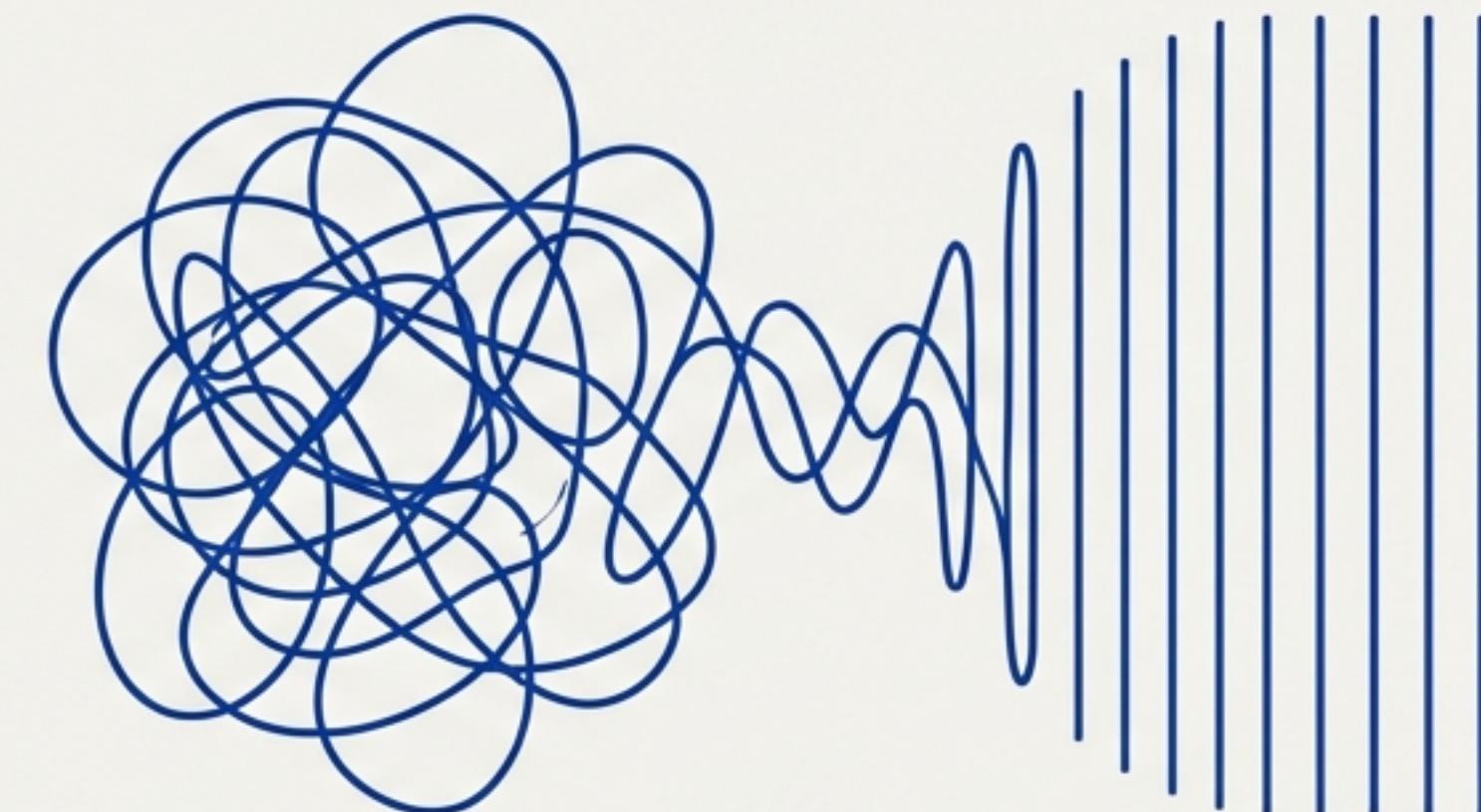
DOMINANDO O COMPILADOR CDS

Um Guia Prático para um Código Robusto e Elegante

O COMPILADOR NÃO É SEU INIMIGO. É O SEU PARCEIRO MAIS RIGOROSO.

Mensagens de erro não são bloqueios; são guias. Elas revelam violações de princípios fundamentais de modelagem em CDS.

Nosso objetivo: aprender a linguagem do **compilador** para construir aplicações mais rápidas, limpas e resilientes.



Vamos transformar mensagens de erro em lições de maestria.

OS TRÊS PILARES DE UM MODELO CDS SÓLIDO



Clareza e Unicidade:

Garantindo uma fonte de verdade única e sem ambiguidades em definições, anotações e extensões.



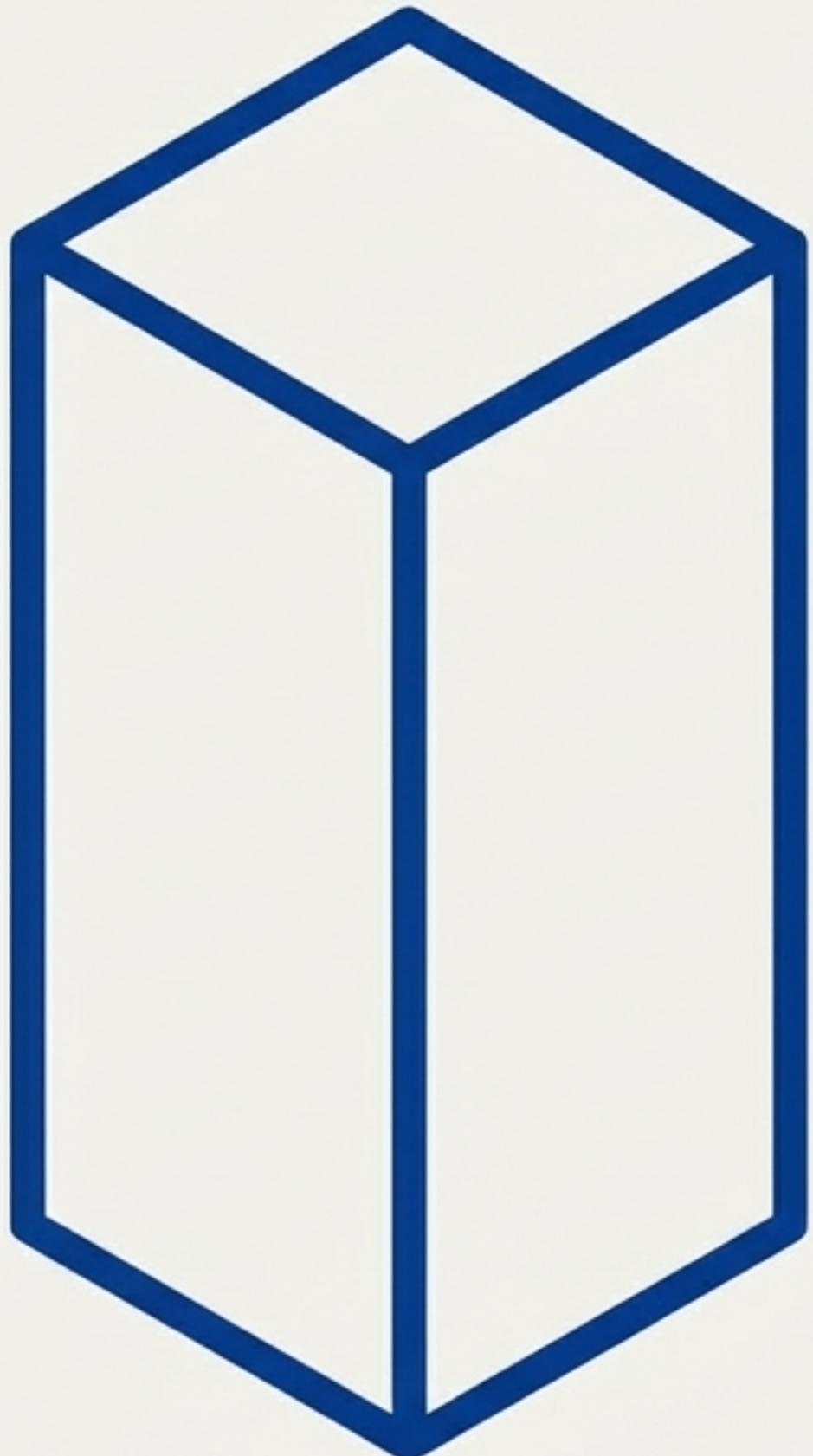
Tipagem Explícita e Segura:

Assegurando que cada dado tenha um tipo bem definido e compreensível pelos backends.



Relações Lógicas e Coerentes:

Dominando a arte das associações e redirecionamentos para um modelo de dados íntegro.



PILAR 1: CLAREZA E UNICIDADE

Um bom modelo não deixa espaço para adivinhação. O compilador exige uma única 'fonte da verdade', seja para uma anotação, uma extensão de entidade ou um nome de artefato. A ambiguidade é a inimiga da estabilidade.

O CONFLITO DA ANOTAÇÃO DUPLICADA (`anno-duplicate-unrelated-layer`)

✗ O Obstáculo

Duas camadas ('layers') não relacionadas ('FooAnnotate.cds', 'BarAnnotate.cds') tentam anotar a mesma entidade 'FooBar'. Como não há uma ordem de dependência definida, o compilador não consegue decidir qual anotação aplicar primeiro.

```
// (1) Base.cds
entity FooBar { }

// (2) FooAnnotate.cds
using from './Base';
annotate FooBar with @Anno: 'Foo';

// (3) BarAnnotate.cds
using from './Base';
annotate FooBar with @Anno: 'Bar';

// (4) All.cds: Combina tudo
using from './FooAnnotate';
using from './BarAnnotate';
```

✓ O Caminho da Maestria

Resolva a ambiguidade estabelecendo precedência. Anotar diretamente no arquivo final ('All.cds') ou criar uma dependência entre as camadas de anotação dá ao compilador uma ordem clara para seguir.

```
// (4) All.cds: Solução com precedência
// (4) All.cds: Solução com precedência
using from './FooAnnotate';
using from './BarAnnotate';

// Esta anotação tem precedência.
annotate FooBar with @Anno: 'Bar';
```

A COLISÃO DA AUTO-EXPOSIÇÃO (`def-duplicate-autoexposed`)

✗ O Obstáculo

Um serviço projeta uma entidade com composições para outras entidades que, apesar de estarem em namespaces diferentes (`ns.first.Foo`, `ns.second.Foo`), têm o mesmo nome. A auto-exposição ignora os namespaces, resultando em uma colisão de nomes no serviço.

```
service ns.MyService {  
    // ✗ Projeção em Base causa colisão  
    // em ns.first.Foo e ns.second.Foo  
    entity BaseView as projection on ns.Base;  
};
```

✓ O Caminho da Maestria

Seja explícito. Em vez de depender da auto-exposição, exponha manualmente as entidades no serviço usando projeções com nomes distintos ou namespaces. Isso remove a ambiguidade e dá a você controle total sobre la API do seu serviço.

```
service ns.MyService {  
    entity BaseView as projection on ns.Base;  
  
    // Expondo manualmente para resolver a colisão  
    entity first.Foo as projection on ns.first.Foo;  
    entity second.Foo as projection on ns.second.Foo;  
};
```

A EXTENSÃO COM ORDEM INSTÁVEL (`extend-unrelated-layer`)

✗ O Obstáculo

A mesma entidade `FooBar` é estendida em múltiplos arquivos (`layers`) que não dependem um do outro. Ao combiná-los, a ordem final dos novos elementos (`foo`, `bar`) torna-se instável e imprevisível.

```
// (2) FooExtend.cds
using from './Base';
```

```
extend FooBar { foo : Integer; }
```

```
// (3) BarExtend.cds
using from './Base';
```

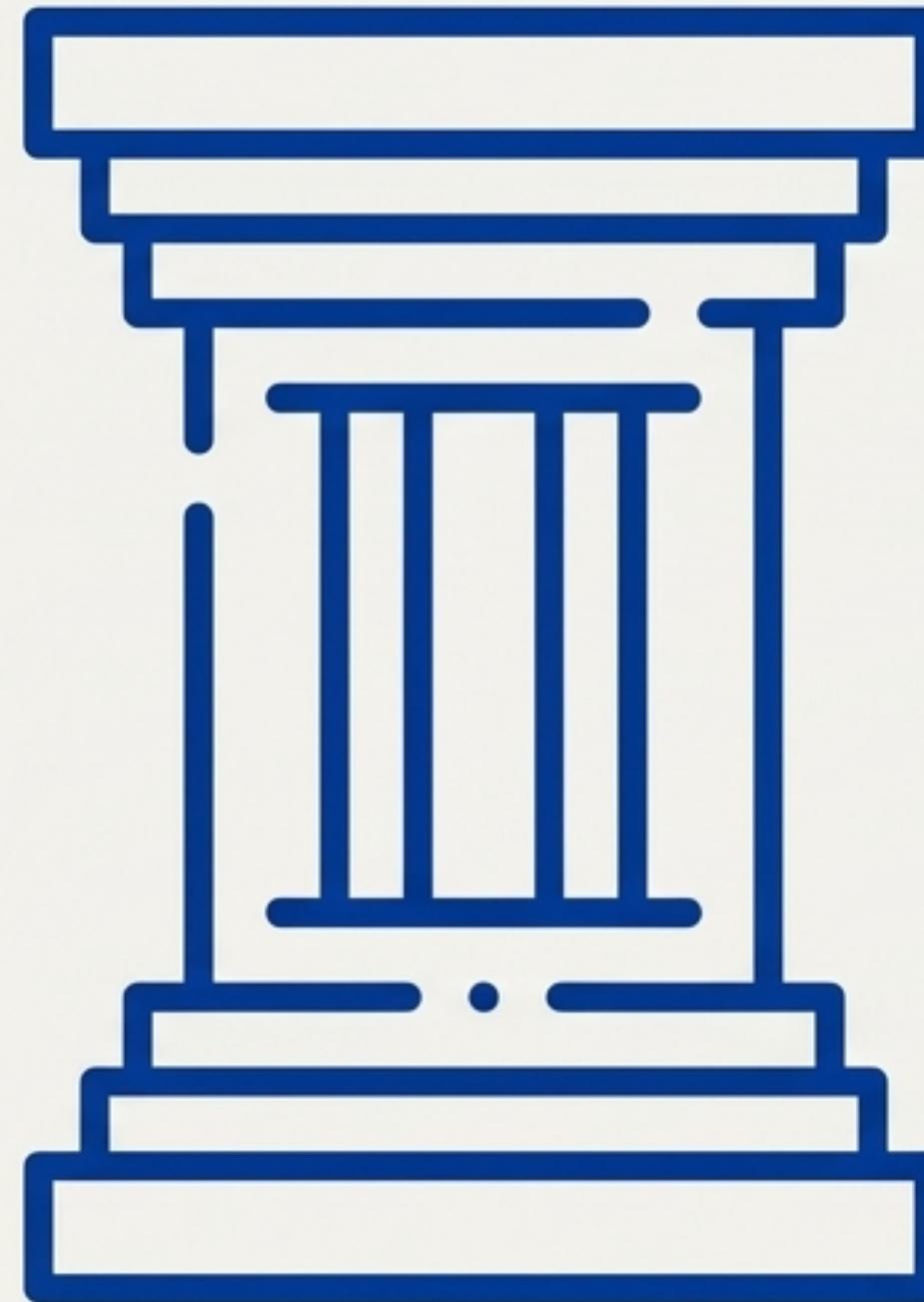
```
extend FooBar { bar : Integer; }
```

✓ O Caminho da Maestria

Consolide as extensões. Mova todas as extensões para a mesma entidade para a mesma camada (geralmente o mesmo arquivo), garantindo uma ordem de elementos estável e previsível.

```
// (2) FooExtend.cds (Consolidado)
using from './Base';
```

```
extend FooBar {
  foo : Integer;
  bar : Integer;
}
```



PILAR 2: TIPAGEM EXPLÍCITA E SEGURA

Dados sem tipo são dados sem significado para os backends (SQL, OData). O compilador nos protege alertando quando um tipo está ausente ou é inferido de forma insegura, garantindo a integridade do nosso modelo de ponta a ponta.

O ELEMENTO SEM TIPO DEFINIDO (`def-missing-type` & `check-proper-type-of`)

✗ O Obstáculo

Um elemento calculado, como `calculatedField` na `ViewFoo`, é definido sem um tipo explícito. Embora o CDS possa permitir, os backends que geram DDL ou metadados de serviço falharão, pois não sabem como interpretar `1+1`.

```
view ViewFoo as select from Foo {  
    1+1 as calculatedField @(anno)  
};  
entity Bar {  
    // ✗ `e` não tem um tipo adequado,  
    // pois `calculatedField` não tem tipo  
    e : ViewFoo:calculatedField;  
};
```

✓ O Caminho da Maestria

Sempre declare seus tipos. Atribua um tipo explícito a todos os elementos, especialmente campos calculados. Isso garante que seu modelo seja traduzível para qualquer tecnologia de persistência ou serviço.

```
view ViewFoo as select from Foo {  
    1+1 as calculatedField @(anno) : Integer  
};
```

O VALOR IMPLÍCITO NO ENUM (`type-missing-enum-value`)

✗ O Obstáculo

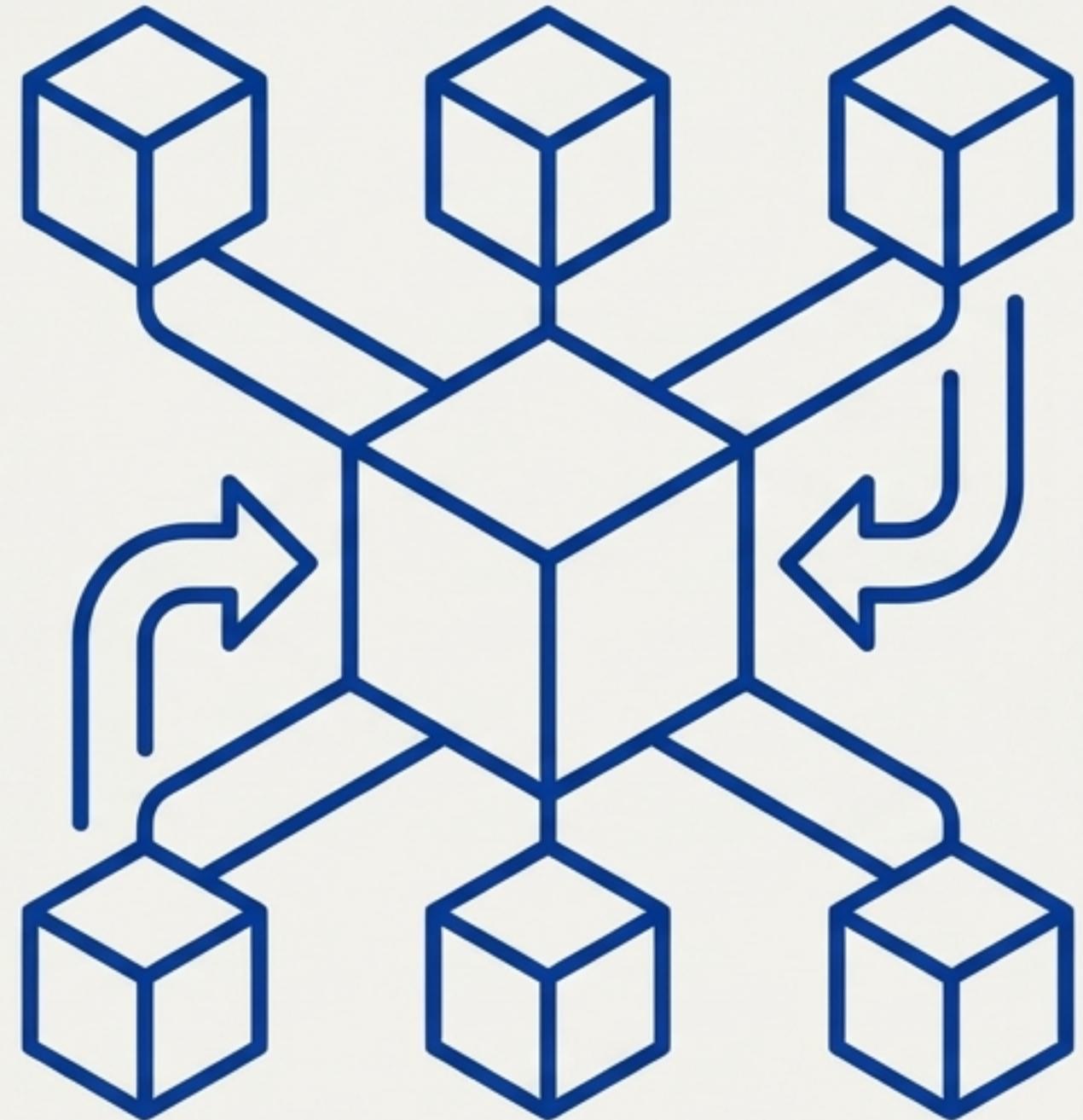
Um `enum` sobre um tipo não-string (como `Integer`) é declarado sem valores explícitos. Isso é perigoso: adicionar um novo membro no futuro poderia reordenar os valores numéricos implícitos e corromper dados existentes. Por exemplo, `Action` poderia mudar de `2` para `3`.

```
category: Integer enum {  
    Fiction; // ✗ valor implícito  
    Action; // ✗ valor implícito  
} default #Action;
```

✓ O Caminho da Maestria

Enums devem ser estáveis. Atribua valores explícitos a cada membro de um enum não-string. Isso garante que os valores persistidos permaneçam consistentes ao longo do tempo, mesmo que o enum evolua.

```
category: Integer enum {  
    Fiction = 1;  
    Action = 2;  
} default #Action;
```



PILAR 3: RELAÇÕES LÓGICAS E COERENTES

Associações são o coração do seu modelo de dados. O redirecionamento (`redirected to`) é uma ferramenta poderosa, mas o compilador garante que seja usado de forma lógica, evitando que as relações apontem para alvos ambíguos, complexos ou sem conexão com a origem.

O REDIRECIONAMENTO SEM CONEXÃO (`redirected-to-unrelated`)

✗ O Obstáculo

Uma projeção tenta redirecionar a associação `toMain` para a entidade `Secondary`. No entanto, `Secondary` não tem nenhuma origem ou conexão com a entidade de destino original da associação (`Main`). O link lógico está quebrado.

```
entity InvalidRedirect as projection on Main {  
    id,  
    // ✗ Redireção inválida para uma entidade  
    // não relacionada  
    toMain: redirected to Secondary,  
};
```

✓ O Caminho da Maestria

Mantenha a linhagem. Redirecione associações apenas para entidades que são, elas mesmas, projeções ou views da entidade de destino original. Neste caso, a solução é redefinir a associação com um `mixin` para criar uma nova relação lógica e explícita.

```
view SecondRedirect as select from FirstRedirect  
mixin {  
    toMain : Association to Main on id = $self.id;  
} into {  
    FirstRedirect.id as id,  
    toMain  
};
```

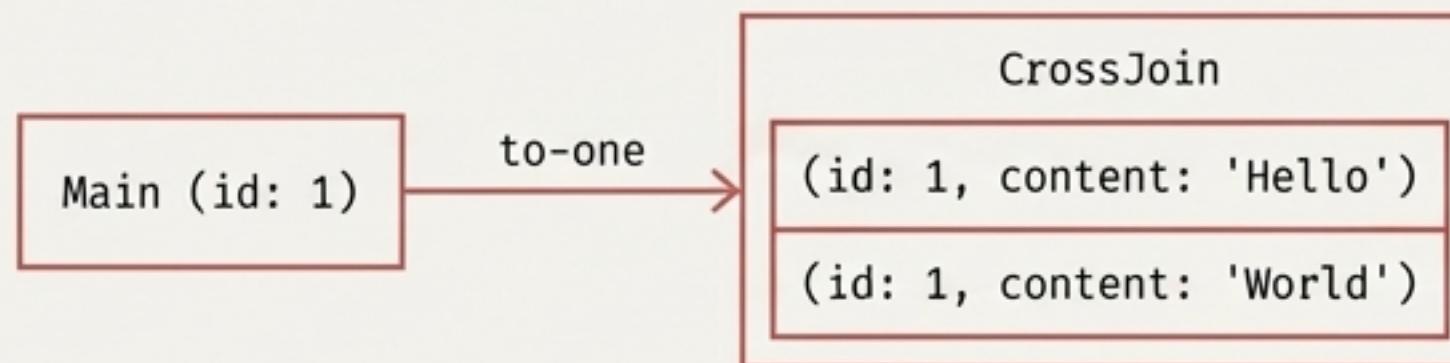
O REDIRECIONAMENTO PARA UM ALVO COMPLEXO (redirected-to-complex)

✗ O Obstáculo

Uma associação `to-one` (`'toMain'`) é redirecionada para uma `view` (`CrossJoin`) que contém um `JOIN` ou `UNION`. Isso pode multiplicar as linhas, transformando sutilmente a associação em `to-many` e quebrando a cardinalidade esperada.

```
entity CrossJoin as SELECT from Main, Secondary;

entity RedirectToComplex as projection on Main {
    id,
    toMain: redirected to CrossJoin, // ✗
};
```



✓ O Caminho da Maestria

Preserve a cardinalidade. Evite redirecionar para views complexas que multiplicam linhas. Se for necessário, adicione uma condição `ON` explícita ou chaves estrangeiras para garantir a unicidade e silenciar o aviso, mas esteja ciente das implicações. O ideal é garantir que o alvo da redireção seja um substituto lógico 1:1.

SUA NOVA MENTALIDADE DE MODELAGEM



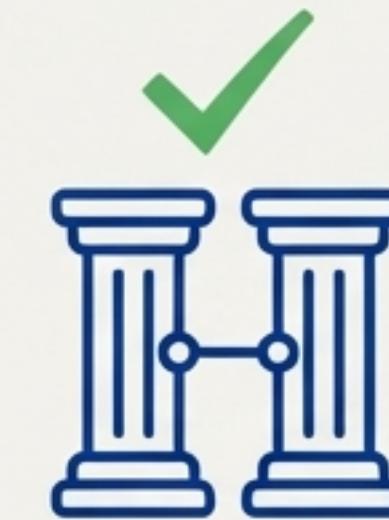
Pilar 1 - Clareza e Unicidade

Pense: "Há apenas uma maneira de interpretar isso?"



Pilar 2 - Tipagem Explícita

Pense: "Cada pedaço de dado pode ser entendido por um banco de dados?"



Pilar 3 - Relações Coerentes

Pense: "Esta associação ainda faz sentido lógico neste novo contexto?"

Adotar esses princípios transforma a programação reativa em design proativo.

PROGRAME COM CONFIANÇA

A maestria em CDS vem de entender sua filosofia. Use o **compilador** como seu guia, aplique estes princípios e construa modelos que são não apenas funcionais, mas também **robustos**, elegantes e prontos para o futuro.