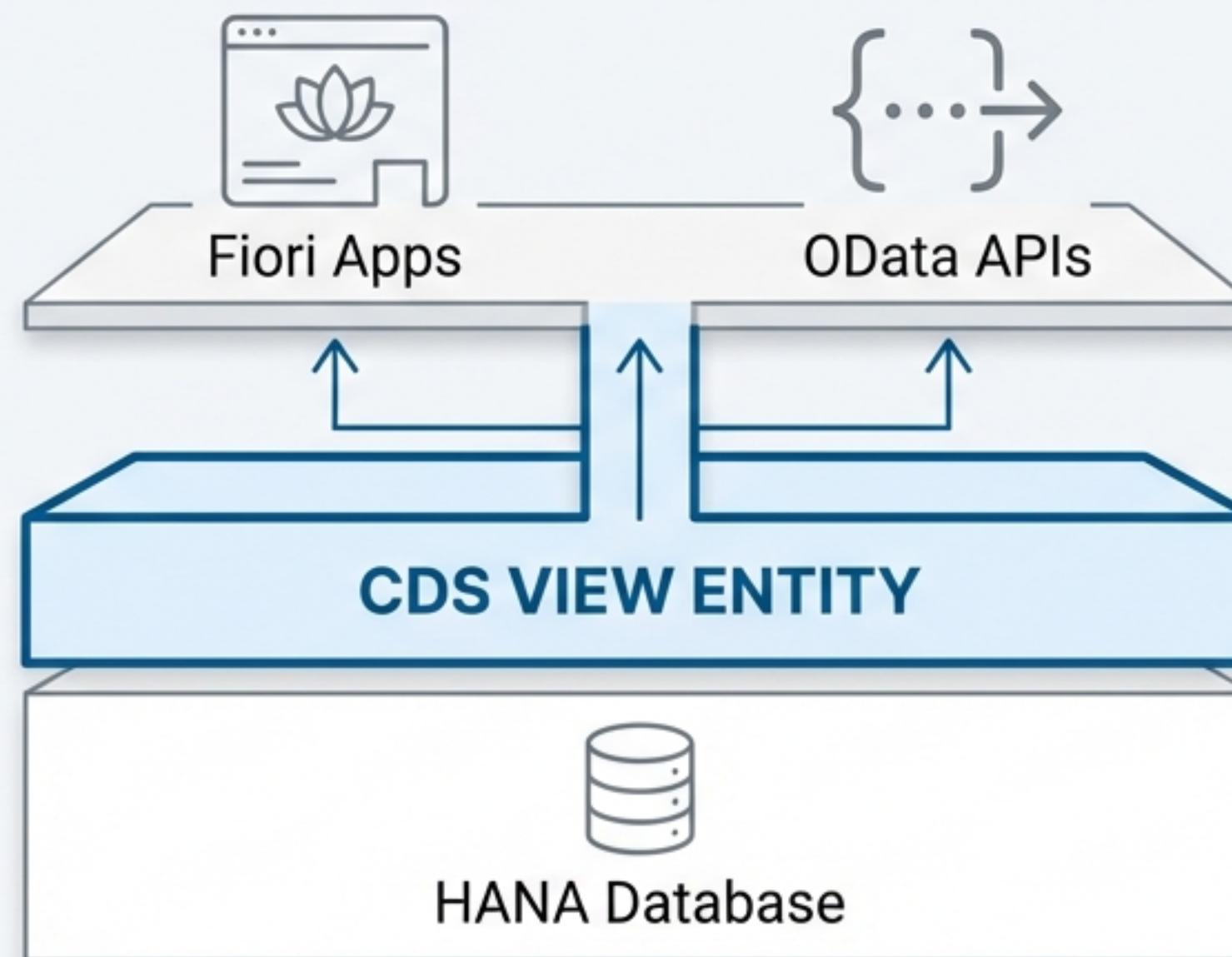


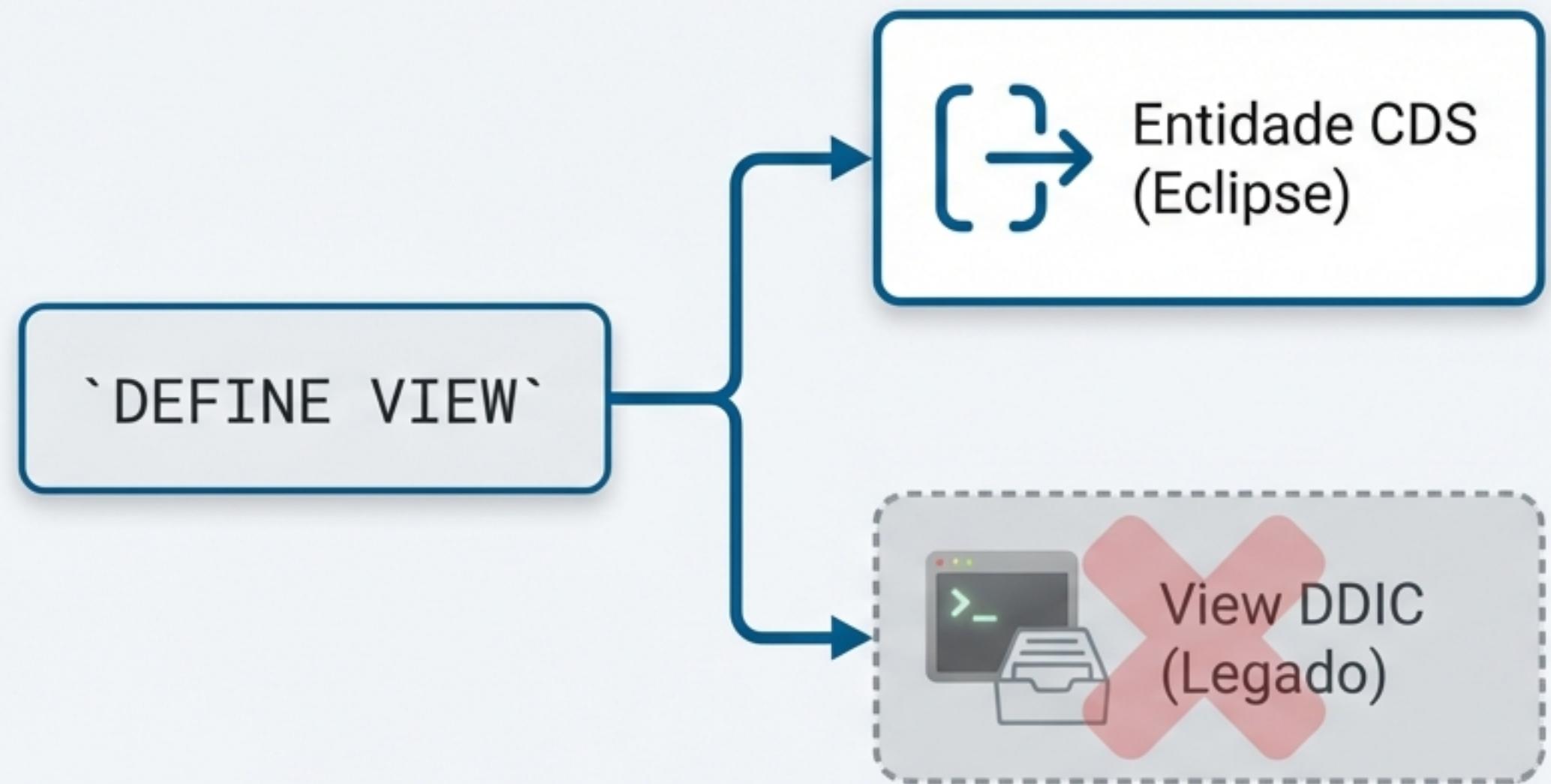
# A Base de Aplicações Modernas: Dominando as CDS Views Essenciais

Em S/4HANA, a modelagem de dados evoluiu. Para construir aplicações Fiori e APIs OData de alta performance com o ABAP RAP, a fundação é uma só: a **CDS View Entity**. Esta não é apenas uma nova sintaxe, é uma nova filosofia de desenvolvimento.



# O Mundo Antigo: O Débito Técnico do `DEFINE VIEW`

Por que a abordagem clássica se tornou um obstáculo?



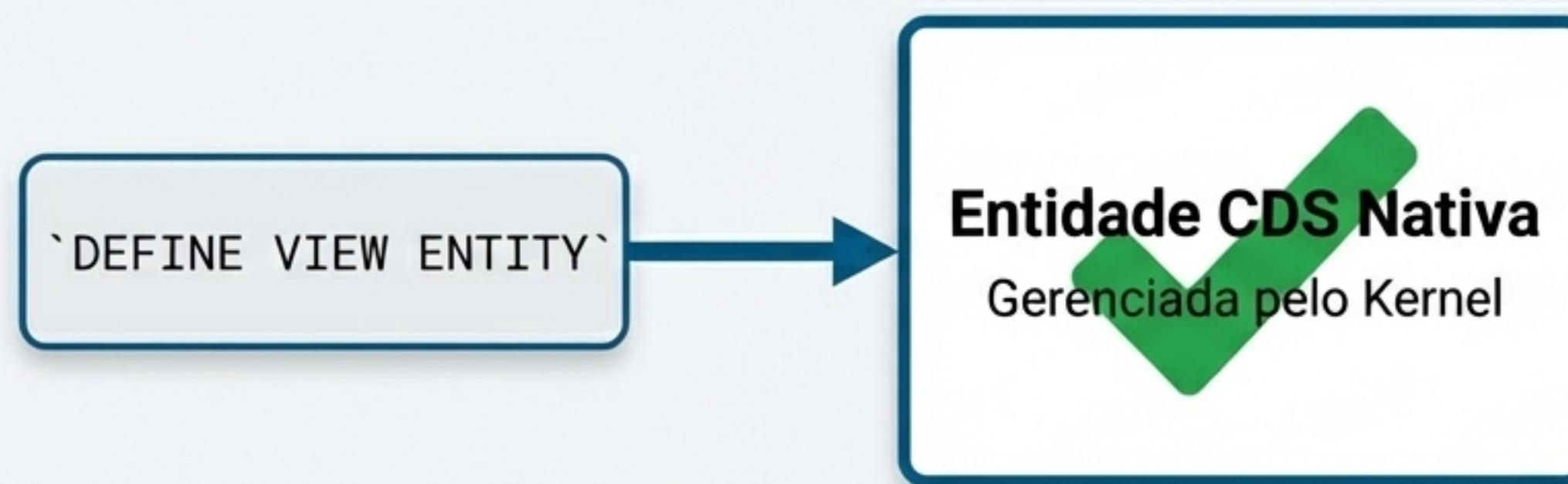
## ! Consequências:

- **Duplicidade:** Geração de objetos desnecessários na SE11.
- **Lentidão:** Ativações mais lentas e complexas.
- **Limitações:** Restrições de namespace (nomes de 16 caracteres) e compatibilidade com regras antigas do Dicionário ABAP.

# A Evolução Necessária:

``DEFINE VIEW ENTITY``

A partir do ABAP 7.55, e como padrão obrigatório no ABAP Cloud, a solução é nativa, limpa e poderosa.



## Principais Vantagens:

- **Artefato Único:** Não cria nenhuma view na SE11. A entidade é gerenciada diretamente pelo Kernel ABAP e pelo banco de dados HANA.
- **Validação Estrita:** O compilador é mais rigoroso, exigindo tipos de dados perfeitos e sem ambiguidades. O resultado é um código mais limpo e seguro.
- **Performance Superior:** Ativação muito mais rápida e um plano de execução otimizado de forma mais eficiente pelo HANA.

# O Confronto Direto: `DEFINE VIEW` vs. `DEFINE VIEW ENTITY`

Característica	‘DEFINE VIEW’ (Abordagem Legada)	‘DEFINE VIEW ENTITY’ (Padrão Moderno)
Artefato na SE11	<b>Sim, cria</b> uma View de Banco de Dados.	<b>Nenhum</b> . Só existe no nível do CDS.
Performance de Ativação	<b>Mais lenta</b> , devido à checagem dupla.	<b>Muito mais rápida</b> e otimizada.
Validação de Sintaxe	Menos rigorosa, tolera ambiguidades.	<b>Estrita</b> . Garante código mais robusto.
Capacidade de Cálculo	Limitada em alguns cenários SQL.	Suporte total a expressões SQL avançadas.
Recomendação de Uso	Apenas para manutenção de código legado.	<b>Sempre usar</b> em S/4HANA e ABAP Cloud.

# A Anatomia de uma View Moderna e Poderosa

Todo artefato `DEFINE VIEW ENTITY` segue uma estrutura clara, dividida em três seções principais.

## A. Anotações de Cabeçalho:

A 'configuração'. Define comportamento, segurança e metadados. Sempre começam com `@`.

\*Exemplo:  
`@AccessControl.authorizationCheck`,  
`@EndUserText.label`\*

## B. Definição e Fonte de Dados:

O 'esqueleto'. Declara o nome da view e sua tabela de origem.

\*Exemplo: `define view entity Z\_I\_TRAVEL  
as select from zrap\_travel`\*

```
// Section A: Annotations  
@Annotation1  
@Annotation2
```

```
// Section B: Definition  
define view entity Z_MY_ENTITY  
as select from my_table
```

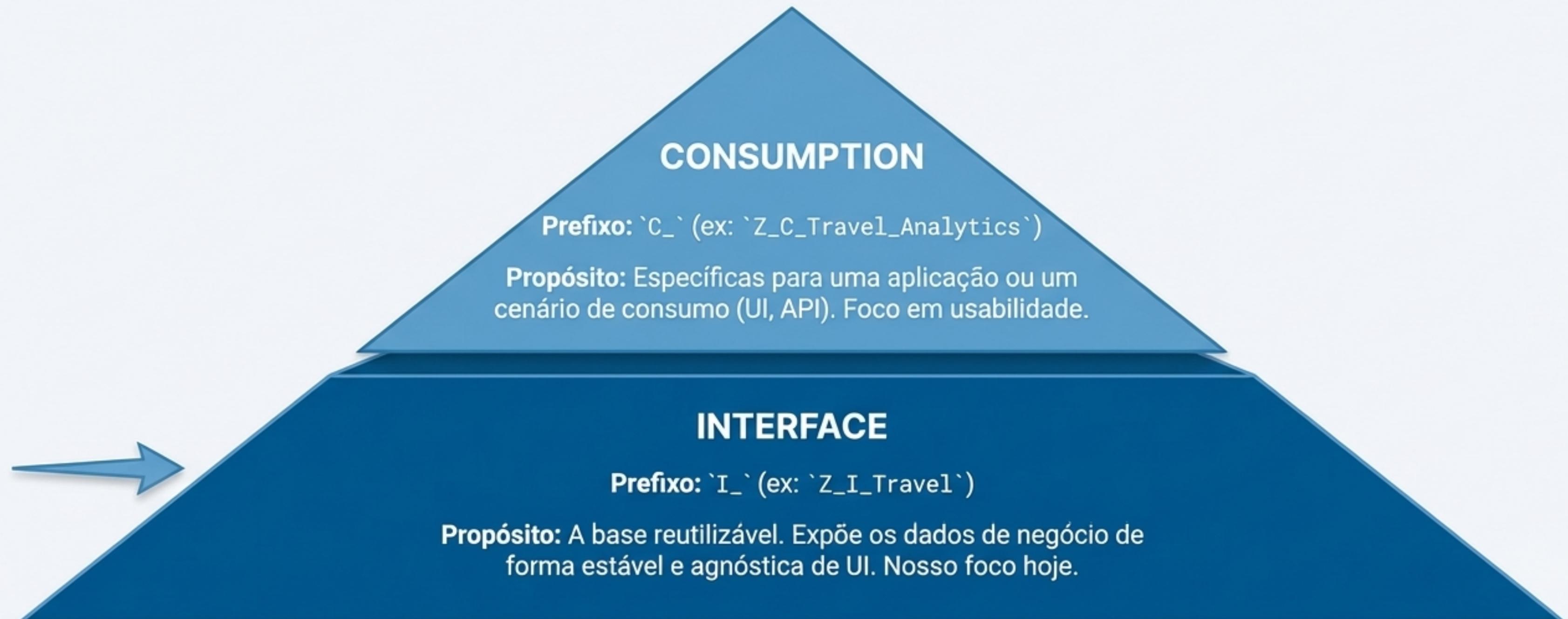
```
// Section C: Projection List  
{  
    field1,  
    field2,  
    ...  
}
```

## C. Lista de Projeção:

O 'coração'. Dentro das chaves `{ }`, selecionamos campos, criamos aliases, fazemos cálculos e definimos associações.

# O Mapa do Território: Entendendo o Virtual Data Model (VDM)

No S/4HANA, as views não são criadas aleatoriamente. Elas seguem o VDM, uma arquitetura em camadas que garante reuso e estabilidade.



# Nosso Exemplo Prático: Construindo a Interface View `Z\_I\_TRAVEL`

Objetivo: Transformar a tabela física `zrap\_travel` em uma interface de negócio limpa, semântica e pronta para consumo.

```
@AccessControl.authorizationCheck: #NOT_REQUIRED
@EndUserText.label: 'Interface View para Viagens'
@Metadata.ignorePropagatedAnnotations: true

define view entity Z_I_TRAVEL
    as select from zrap_travel as Travel
{
    key travel_uuid           as TravelUUID,
    travel_id                 as TravelID,
    agency_id                 as AgencyID,
    customer_id               as CustomerID,
    begin_date                as BeginDate,
    end_date                  as EndDate,
    @Semantics.amount.currencyCode: 'CurrencyCode'
    booking_fee               as BookingFee,
    @Semantics.amount.currencyCode: 'CurrencyCode'
    total_price                as TotalPrice,
    currency_code              as CurrencyCode,
    descrption                 as Description,
    overall_status              as OverallStatus,
    /* --- Campos de Auditoria (Admin Data) --- */
    @Semantics.user.createdBy: true
    created_by                 as CreatedBy,
    @Semantics.systemDateTime.createdAt: true
    created_at                  as CreatedAt,
    @Semantics.user.lastChangedBy: true
    last_changed_by             as LastChangedBy,
    @Semantics.systemDateTime.lastChangedAt: true
    last_changed_at             as LastChangedAt
}
```

# Fundamentos da Projeção: Chaves e a Arte do `CamelCase`

## A Palavra-Chave `key`

```
{  
  key travel_uuid as TravelUUID,  
  ...  
}
```

## O Poder do Alias `as`

```
{  
  ...  
  travel_id as TravelID,  
  ...  
}
```

- ✓ **O que é:** `key travel\_uuid as TravelUUID`
- ✓ **Por que é Crítico:** Define o identificador único do registro. Sem isso, frameworks OData e Fiori Elements não conseguem executar operações de leitura, edição ou navegação em itens individuais. Esquecer a chave `key` quebra a aplicação.

- ✓ **O que é:** `travel\_id as TravelID`
- ✓ **Por que é Estratégico:** Converte nomes técnicos do banco (TRAVEL\_ID) para o padrão **CamelCase** (TravelID). Interfaces web modernas (UI5, Fiori, APIs REST) utilizam este padrão. Resolvemos a padronização na fonte, facilitando a vida do desenvolvedor frontend.

# O Segredo da Automação (Parte 1): A Semântica na Interface

Anotações que ensinam o sistema sobre o significado dos seus dados.

Foco na Anotação:

```
@Semantics.amount.currencyCode: 'CurrencyCode'  
total_price as TotalPrice
```

O que ela faz: Vincula um campo de valor (`TotalPrice`) ao seu campo de moeda correspondente (`CurrencyCode`).

## O Impacto 'Mágico' na UI:

Sem a Anotação ×

TotalPrice
1000
1500.50
200

Com a Anotação ✓

TotalPrice
<b>1.000,00 EUR</b>
<b>1.500,50 USD</b>
<b>200 JPY</b>

# O Segredo da Automação (Parte 2): Delegando Lógica ao Framework RAP

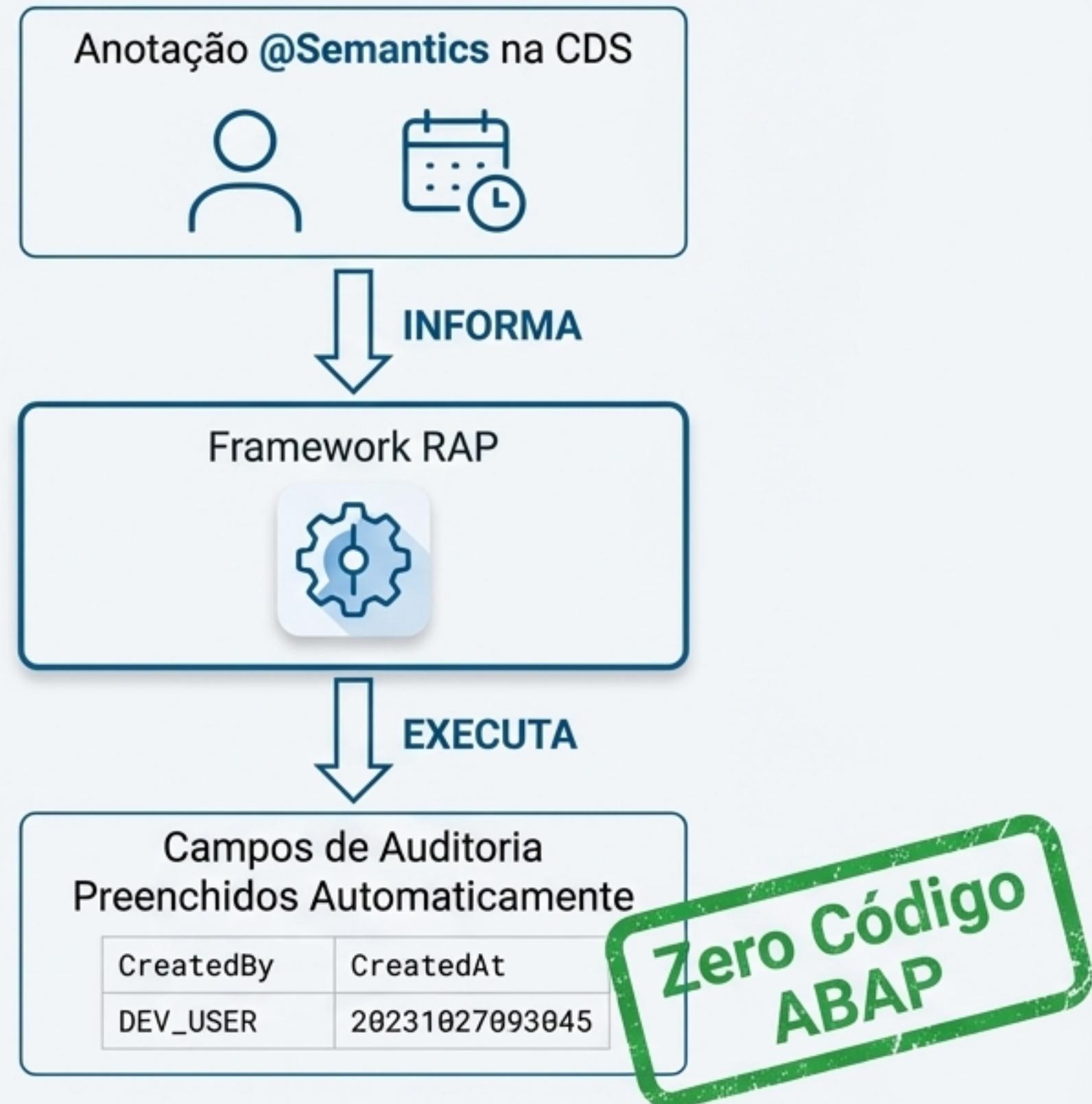
## Foco nas Anotações de Auditoria:

```
@Semantics.user.createdBy: true  
@Semantics.systemDateTime.createdAt: true
```

- ✓ O que elas fazem: Marcam os campos com um significado de negócio específico: "Usuário de Criação" e 'Data/Hora de Criação'.

## O Impacto 'Mágico' no Backend (Em um cenário RAP Managed):

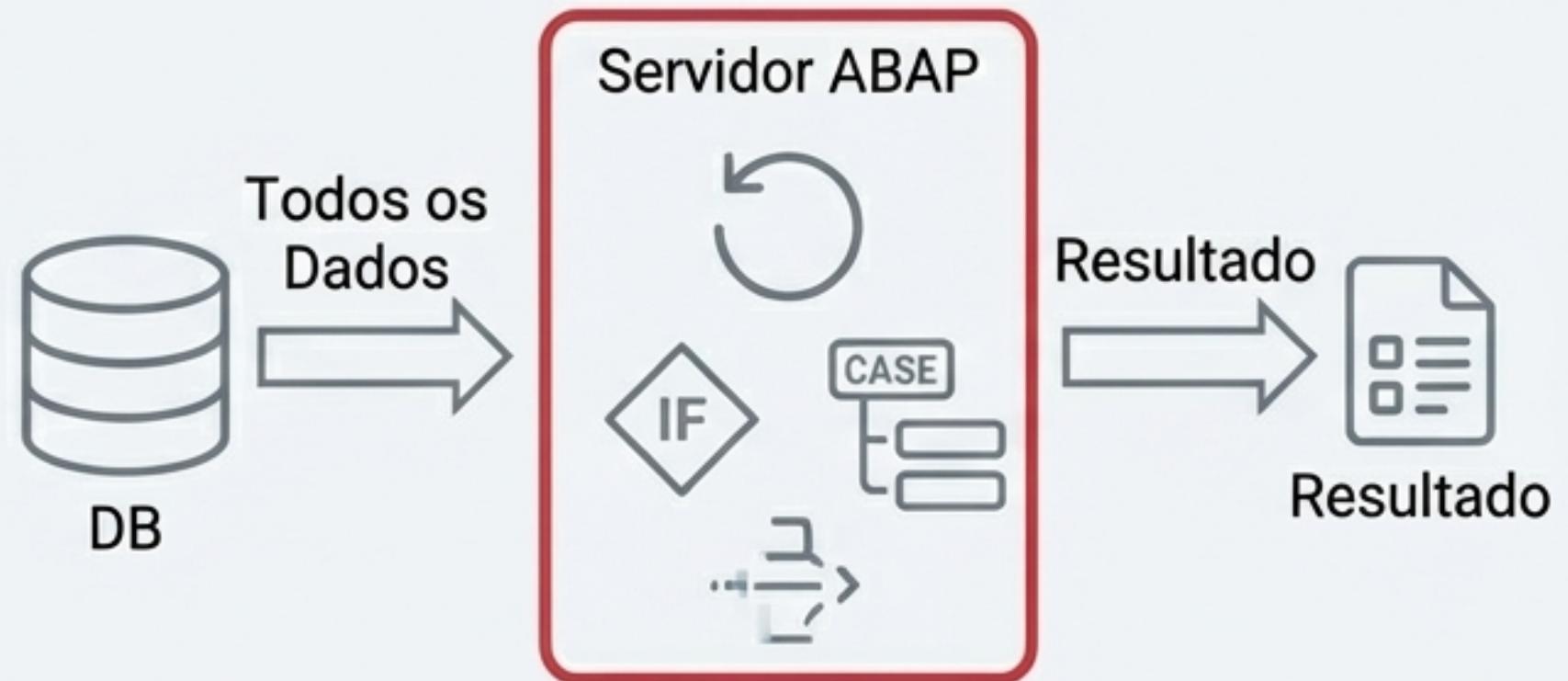
- ✓ O framework RAP **identifica** essas anotações.
- ✓ No momento de um 'INSERT', ele **preenche automaticamente** os campos com `sy-uname` e `sy-timestamp`.
- ✓ **Resultado:** O desenvolvedor não escreve **uma única linha de código ABAP** para implementar a lógica de auditoria.



# O Segredo da Performance: Deixando o HANA Fazer o Trabalho Pesado

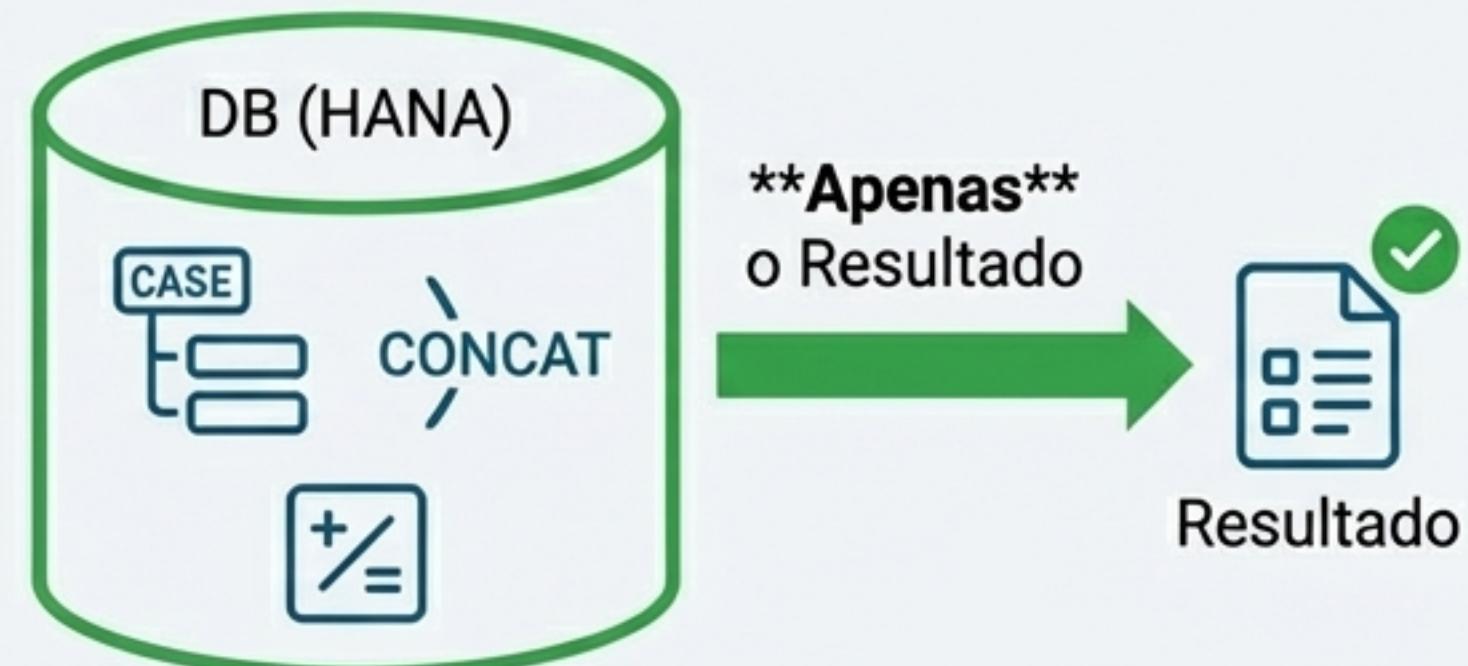
## O Paradigma Antigo

1. `SELECT \*` da tabela para uma tabela interna.
2. `LOOP AT itab` no servidor ABAP para fazer cálculos, concatenações ou conversões.
3. Ineficiente para grandes volumes de dados.



## O Paradigma Moderno (Code Pushdown)

1. Incorporamos os cálculos **diretamente na definição da CDS View**.
2. O otimizador do HANA executa essas operações de forma massivamente paralela, diretamente nos dados.
3. O servidor ABAP recebe apenas o resultado final.



# Code Pushdown na Prática: SQL Moderno em Ação

Exemplos de lógica que você pode mover para dentro da sua View.

## Lógica Condisional (CASE)



Para categorizar dados na fonte.

```
case
  when total_price > 1000 then 'High
Value'
  when total_price > 500 then 'Medium
Value'
  else 'Low Value'
end as PriceCategory
```

## Operações de String e Data



Para transformar dados brutos.

```
concat_with_space(first_name,
last_name, 1) as FullName
dat_s_days_between(begin_date,
end_date) as DurationDays
```

## Conversão de Tipo (Casting)



Para garantir a compatibilidade de tipos.

```
cast(total_price as abap.fltp) as
PriceAsFloat
```

# O Glossário do Desenvolvedor Moderno



## CDS View Entity

A evolução da CDS View. Entidade gerenciada pelo kernel ABAP, sem artefato na SE11, com performance e validação superiores.



## Alias (`as`)

Nome alternativo para um campo na projeção, usado para converter nomes técnicos em nomes semânticos (CamelCase).



## CamelCase

Padrão de nomenclatura (`ExemploDeNome`) essencial para consumo por UIs e APIs modernas.



## Key

Palavra-chave que define os campos de identificação única. Mandatória para o funcionamento correto de serviços OData.



## @Semantics

Família de anotações que descreve o significado de um dado, habilitando automações no framework RAP e na renderização do Fiori.

# Os Princípios da Modelagem de Dados Moderna

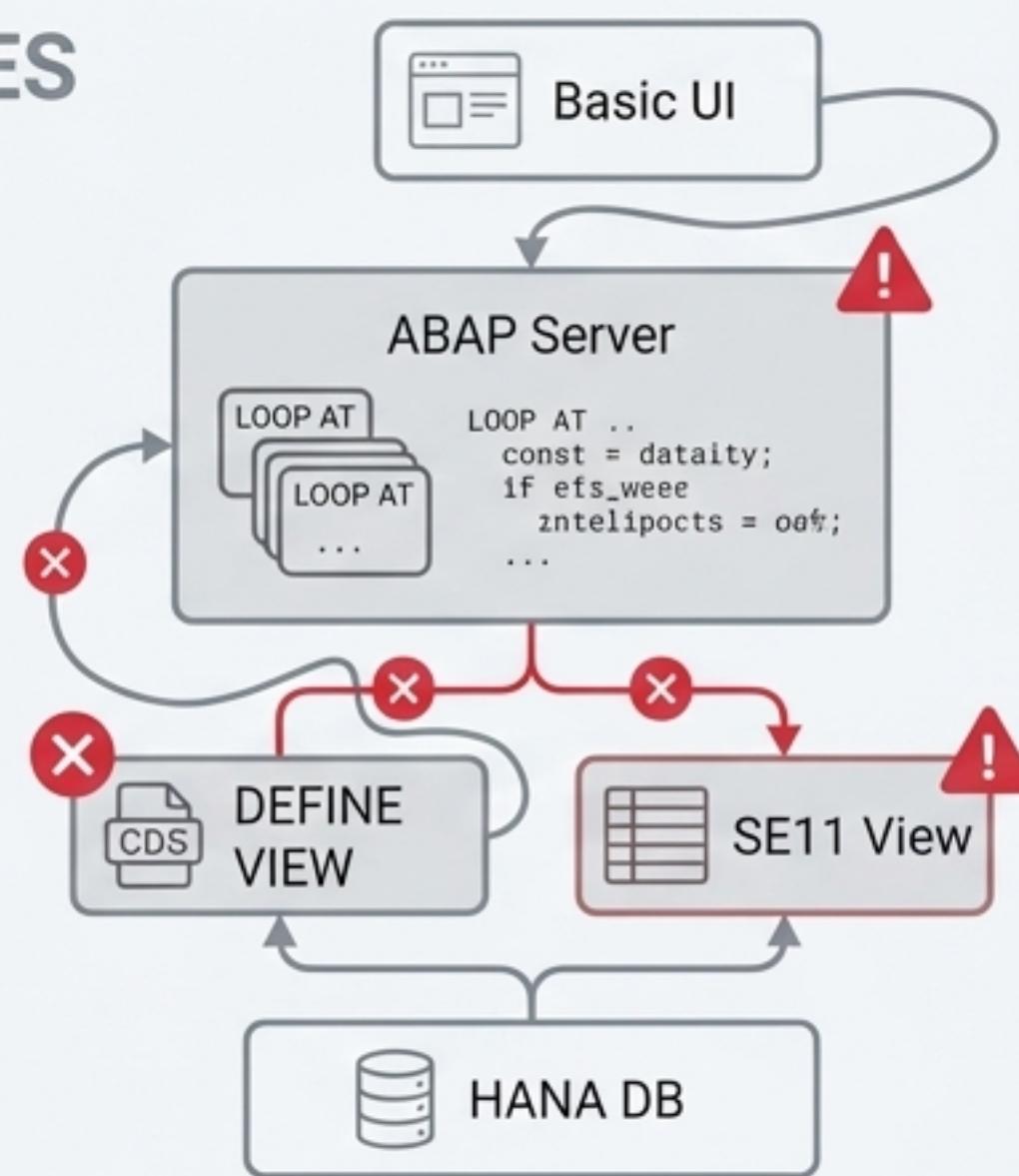
Para garantir código robusto, performático e preparado para o futuro, siga estas regras de ouro.

-  **1** **SEMPRE** use `DEFINE VIEW ENTITY`. A abordagem `DEFINE VIEW` é estritamente para código legado.
-  **2** **PENSE** em `CamelCase`. Seus aliases são a API que você expõe para o mundo. Faça-a limpa e padronizada.
-  **3** **NUNCA** se esqueça da `key`. É a base para que os frameworks OData e Fiori funcionem corretamente.
-  **4** **DELEGUE** trabalho com `@Semantics`. Deixe o framework cuidar da formatação de UI e da lógica de auditoria.
-  **5** **EMPURRE** a lógica para o HANA\*\*. Use cálculos na view para evitar loops ineficientes no código ABAP.

# Mais que Sintaxe, Uma Nova Arquitetura

Adotar `DEFINE VIEW ENTITY` não é apenas aprender um novo comando. É abraçar uma arquitetura mais limpa, integrada e performática, onde o m modelo de dados é o coração inteligente da aplicação.

ANTES



AGORA

