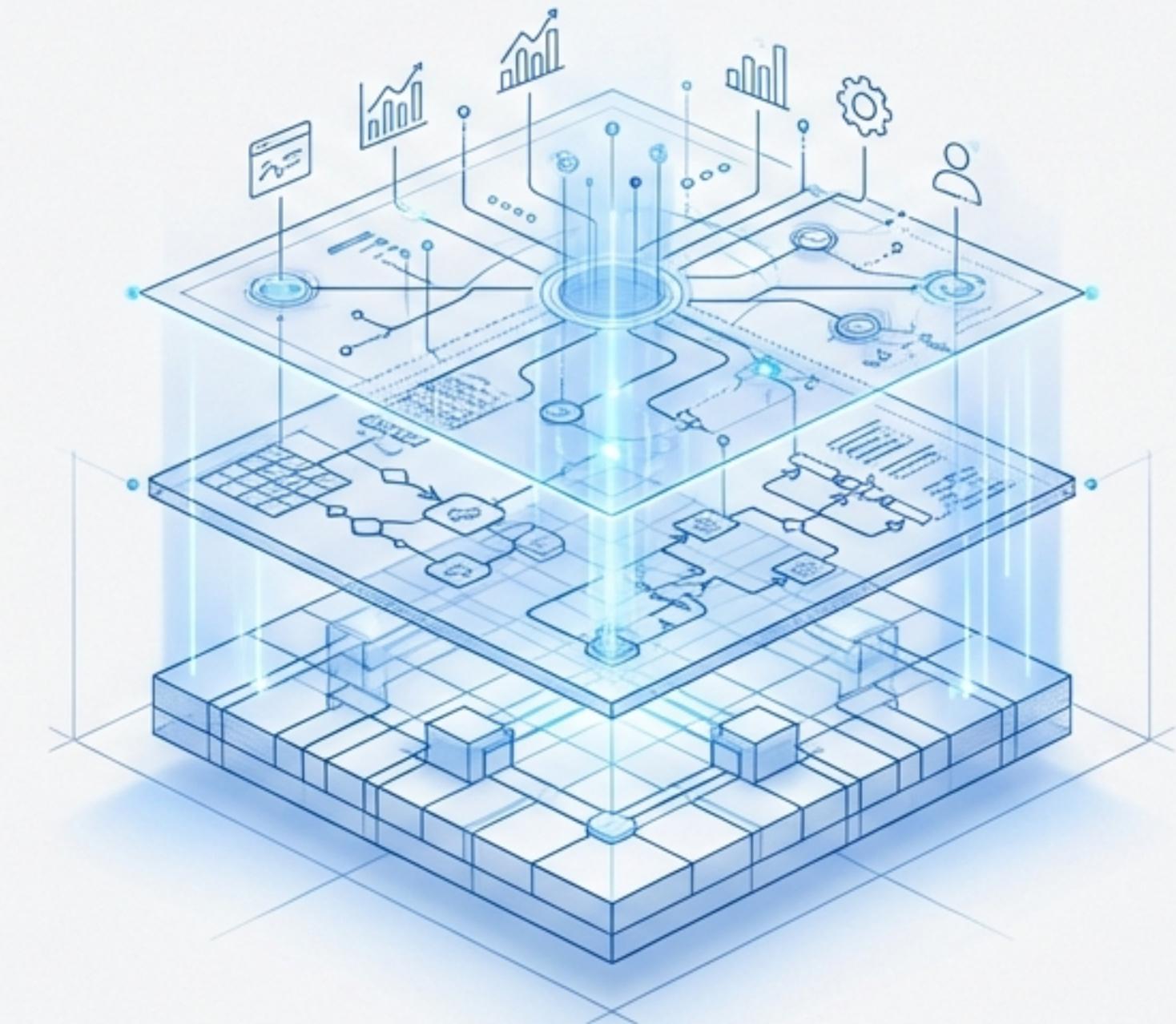


Modelagem de Dados Moderna em ABAP

Da Fundação Física à Interface Inteligente com ABAP CDS



Uma jornada construtiva através do ABAP RESTful Application Programming Model (RAP).

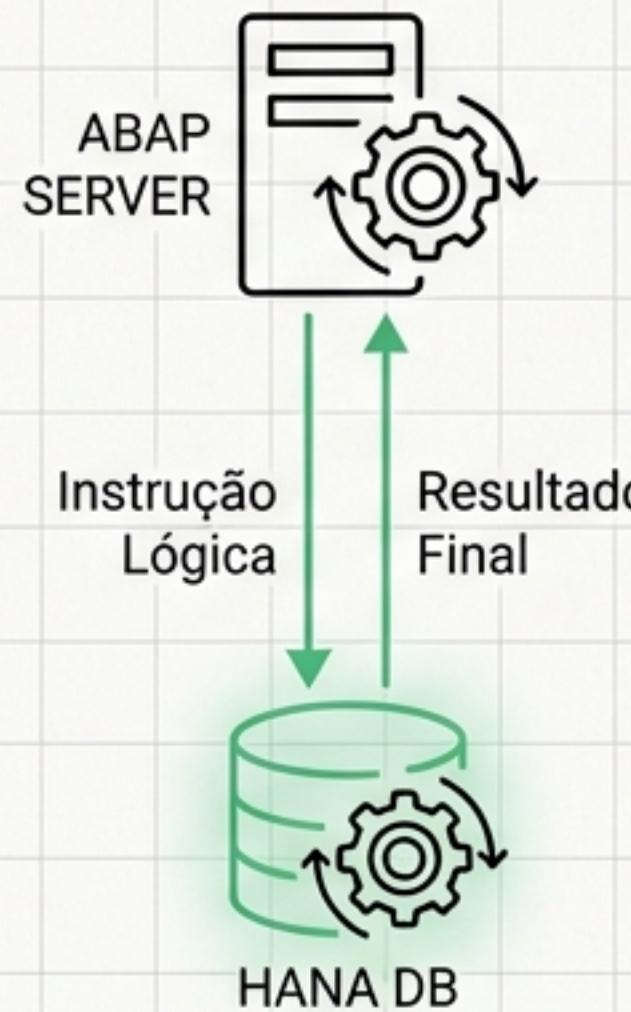
A Revolução do "Code Pushdown": Por Que Tudo Mudou

O Paradigma Antigo: "Data-to-Code"



O ABAP seleciona milhões de linhas, transfere pela rede e processa em tabelas internas gigantes. Ineficiente e lento.

O Novo Paradigma: "Code-to-Data"



O ABAP envia a lógica para o HANA. O banco processa, agrupa e devolve apenas o resultado final. Performance massiva.

Deixamos de trazer os dados até o código.
Agora, levamos o código até os dados.

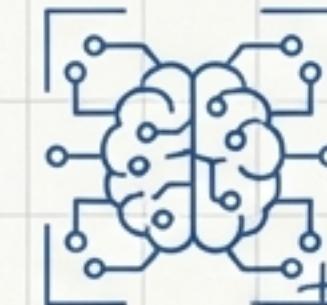
A Nova Caixa de Ferramentas: A Divisão de Responsabilidades



ABAP Dictionary (via ADT)

Persistência Física e Semântica Técnica.

- **Tabelas de Banco de Dados:** Onde os dados realmente moram (DEFINE TABLE).
- **Elementos de Dados / Domínios:** Tipos reutilizáveis, labels de tela e ajudas de pesquisa.



ABAP Core Data Services (CDS)

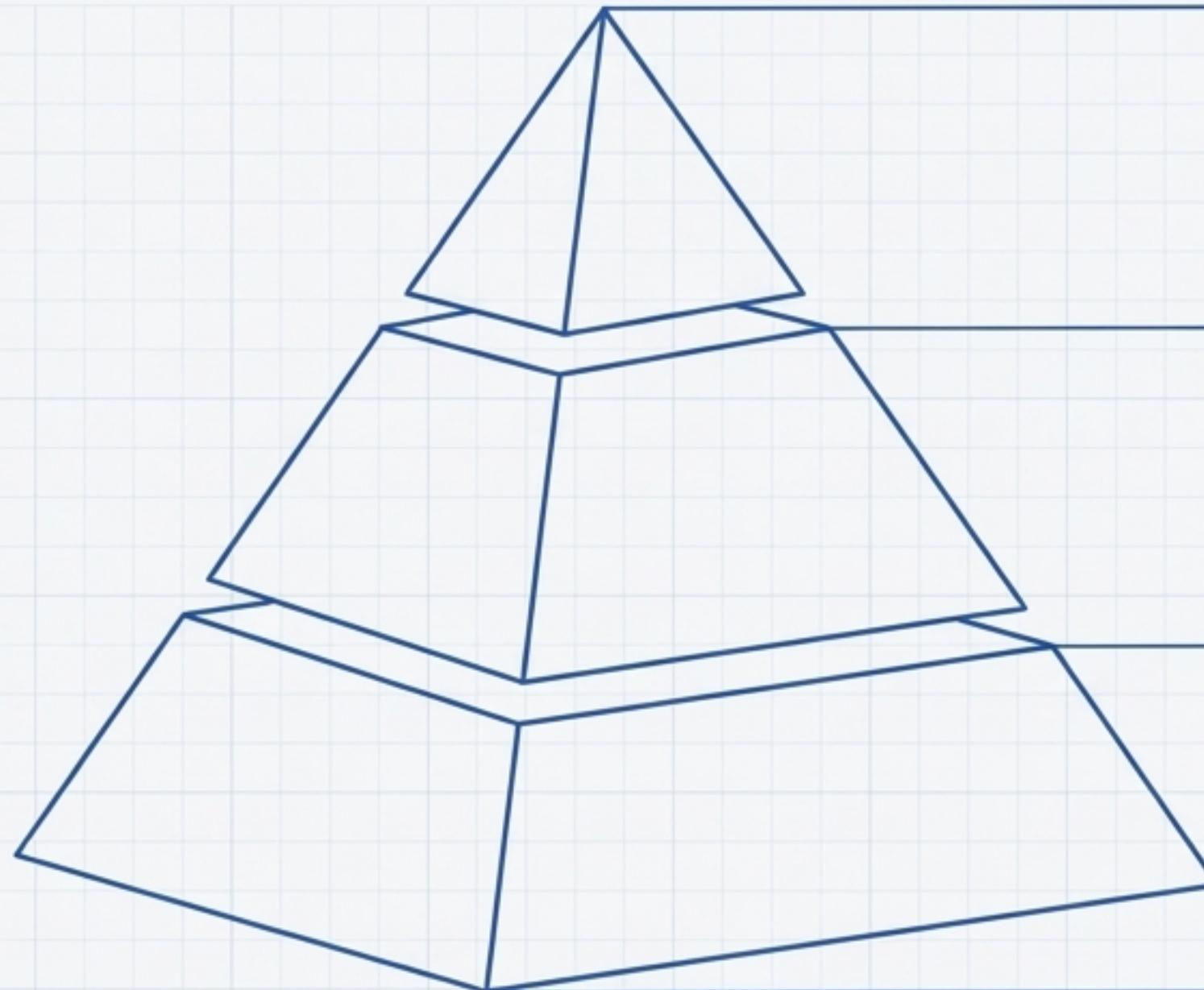
Modelagem Lógica e Enriquecimento Semântico.

- **Views (Visões):** Lógica de leitura, Associações (ASSOCIATION), Cálculos (CASE), Agregações (SUM).
- **Controle de Acesso (DCLs):** Segurança em nível de linha.
- **Metadados de UI (MDEs):** Anotações para Fiori Elements.

O Dicionário define **onde** e **como** os dados são armazenados.
O CDS define **o que** eles significam e **como** devem ser lidos e apresentados.

A Planta do Projeto: A Arquitetura-Guia do Virtual Data Model (VDM)

No S/4HANA, desencorajamos o acesso direto a tabelas físicas. Em vez disso, construímos uma **hierarquia de CDS Views** (o **VDM**) para criar uma camada **de abstração** que protege a aplicação de mudanças no banco e fornece **visões de negócio prontas** para consumo.



CONSUMPTION / PROJECTION VIEWS

Prefixo: `C_` (ex: `C_ApproveTravel`)

Anotação: @VDM.viewType: #CONSUMPTION

Propósito: A ponta do iceberg. Específica para uma UI ou API. Contém anotações @UI. Baixa reutilização.



COMPOSITE VIEWS

Prefixo: `I_` (ex: `I_SalesOrderWithCustomer`)

Anotação: @VDM.viewType: #COMPOSITE

Propósito: Combinam Interface Views. Contêm a lógica de negócios reutilizável. A base para a analítica.



BASIC / INTERFACE VIEWS

Prefixo: `I_` (ex: `I_Product`)

Anotação: @VDM.viewType: #BASIC

Propósito: A fonte da verdade. Espelham os dados brutos da tabela física, mas com nomes limpos e semânticos. Alta reutilização.



Vamos construir a nossa aplicação seguindo esta planta, camada por camada.

Passo 1: Lançando os Alicerces com `DEFINE TABLE`

```
@EndUserText.label : 'Tabela de Viagens - RAP Demo'  
@AbapCatalog.tableCategory : #TRANSPARENT  
@AbapCatalog.deliveryClass : #A  
@AbapCatalog.dataMaintenance : #RESTRICTED  
define table zrap_travel {  
    key client      : abap.clnt not null;  
    key travel_uuid : sysuuid_x16 not null; // A chave moderna  
  
    travel_id       : /dmo/travel_id;  
    agency_id       : /dmo/agency_id;  
    customer_id     : /dmo/customer_id;  
  
    @Semantics.amount.currencyCode : 'zrap_travel.currency_code'  
    total_price     : /dmo/total_price;  
    currency_code   : /dmo/currency_code;  
  
    // Campos de auditoria gerenciados pelo RAP  
    created_by      : syuname;  
    created_at      : timestamppl;  
    last_changed_by : syuname;  
    last_changed_at : timestamppl;  
}
```



Insight de Arquiteto: Por que usar `sysuuid_x16` como chave?

UUIDs (Identificadores Únicos Universais) são a chave primária padrão no RAP. Eles permitem a implementação de cenários complexos como **Drafts (Rascunhos)**, onde um registro precisa existir com uma chave única antes de ser validado e numerado oficialmente (**Late Numbering**).

O `travel_id` legível torna-se um campo secundário de busca.

Adeus, SE11. Olá, abapGit! A definição como código permite versionamento, comparação (diff) e produtividade.

Passo 2: Definindo os Blocos de Construção Semânticos



Nível 2 (Topo) - ELEMENTO DE DADOS

Responsabilidad: Adiciona o significado de NEGÓCIO.

**Field Labels (a mágica do Fiori):

- Short (10): 'Tel. Com.'
- Medium (20): 'Telefone Comercial'
- Long (40): 'Número do Telefone Comercial'
- Heading: 'Telefone'
- Ajuda F1: 'Insira o telefone principal do contato.'



Nível 1 (Base) - DOMÍNIO

Responsabilidad: Define as propriedades TÉCNICAS.

Exemplo:

- Tipo de Dado: 'CHAR30'
- Permite Minúsculas: 'true'
- Rotina de Conversão: 'ALPHA'

Investir tempo aqui se paga sozinho. O Fiori Elements é responsivo: ele herda e escolhe automaticamente o label correto (Longo, Médio ou Curto) com base no tamanho da tela, garantindo uma UI perfeita em qualquer dispositivo e em todos os idiomas.

Passo 3: A Primeira Camada de Proteção - Interface Views (Basic)

Esta é a primeira camada do nosso VDM. Ela não contém filtros de negócio para garantir máxima reutilização. Seu objetivo é esconder os detalhes da tabela física e **criar uma interface estável e amigável para APIs e outras views**, usando nomes em **CamelCase**.

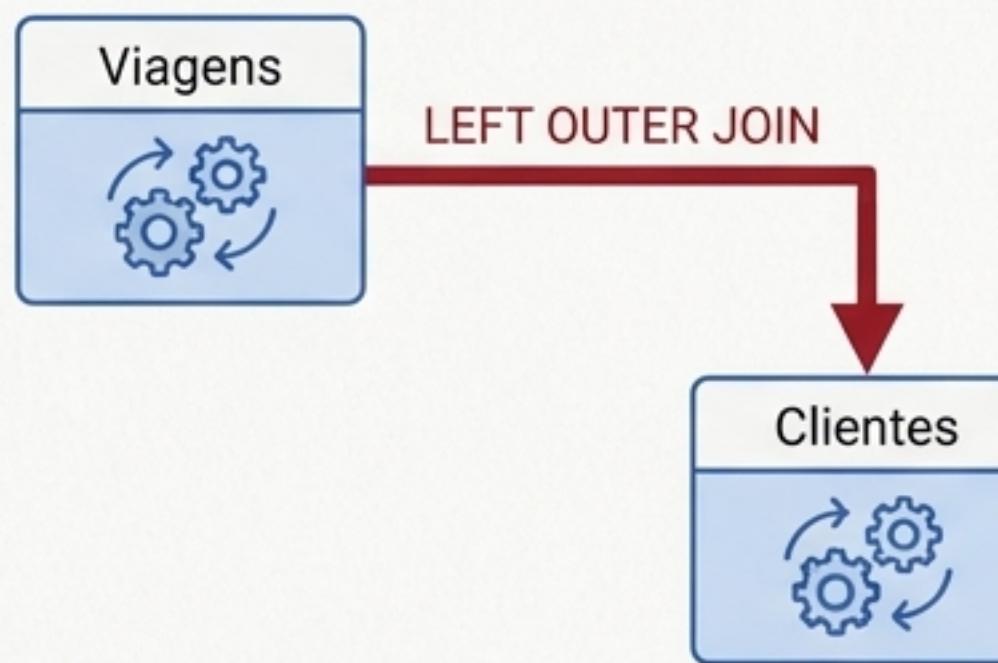
```
@AccessControl.authorizationCheck: #NOT_REQUIRED
@EndUserText.label: 'Interface View para Viagens'
define view entity Z_I_TRAVEL
  as select from zrap_travel as Travel
  {
    key travel_uuid      as TravelUUID, // Alias para CamelCase
    travel_id           as TravelID,
    agency_id           as AgencyID,
    customer_id         as CustomerID,
    begin_date          as BeginDate,
    end_date             as EndDate,
    @Semantics.amount.currencyCode: 'CurrencyCode'
    total_price          as TotalPrice,
    currency_code        as CurrencyCode,
    description          as Description,
    overall_status       as OverallStatus
  }
```

Estamos transformando nomes técnicos como `travel_uuid` em nomes amigáveis como `TravelUUID`. Interfaces modernas (Fiori, APIs REST) padronizam o uso de CamelCase. O CDS resolve isso na fonte.

Passo 4: Conectando as Estruturas com Associações

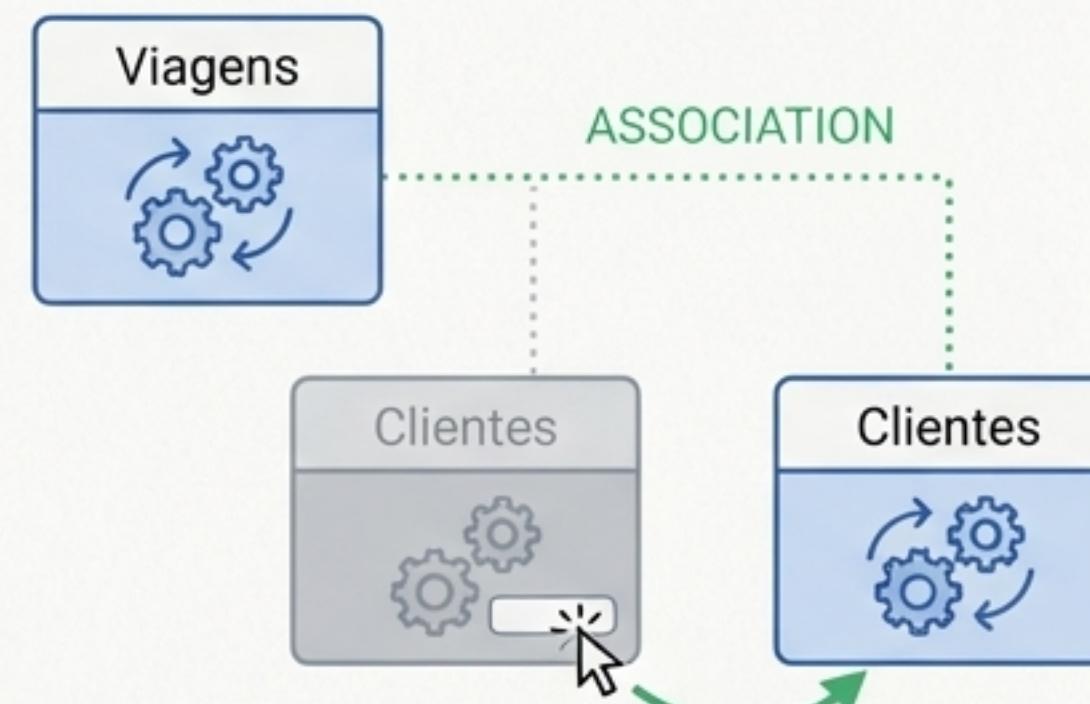
O poder do 'Join sob Demanda'

'JOIN' (Eager Loading)



O 'JOIN' é executado fisicamente **toda vez** que a view é lida, mesmo que você só precise de campos da tabela de Viagens. Desperdício de recursos.

'ASSOCIATION' (Lazy Loading)

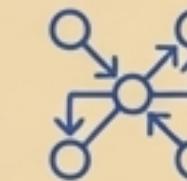


A Associação é um relacionamento lógico. O 'JOIN' físico só acontece no momento em que um campo da entidade associada é explicitamente solicitado. Performance e modelos mais limpos.

As Associações são a base da navegação no Fiori e permitem criar modelos ricos sem penalizar a performance de consultas simples.

Aplicando Associações e Expondo Relacionamentos

```
define view entity Z_I_TRAVEL
  as select from zrap_travel as Travel
  // Definição das associações (relacionamentos lógicos)
  association [0..1] to /DMO/I_Agency as _Agency
    on $projection.AgencyID = _Agency.AgencyID
  association [0..1] to /DMO/I_Customer as _Customer
    on $projection.CustomerID = _Customer.CustomerID
  association [0..1] to I_Currency as _Currency
    on $projection.CurrencyCode = _Currency.Currency
{
  key travel_uuid as TravelUUID,
  agency_id as AgencyID,
  customer_id as CustomerID,
  ...
  /* --- Exposição das Associações (Publicação) --- */
  // Ao incluir o alias aqui, transformamos o relacionamento
  // "privado" em uma "Navegação Pública" para OData.
  _Agency,
  _Customer,
  _Currency
}
```



Insight de Arquiteto: Cardinalidade e Exposição

A **Cardinalidade** (`[0..1], `[1..*]`) não é apenas documentação; ela informa ao otimizador do HANA se pode usar um `INNER JOIN` (mais rápido) ou deve usar um `LEFT OUTER JOIN`.

Expor a associação (colocar `_Alias` na lista de campos) a transforma em uma 'Navigation Property' no serviço OData, permitindo que o Fiori navegue entre entidades (drill-down) automaticamente.

Passo 5: Adicionando Inteligência com Expressões SQL

Em vez de fazer loops em ABAP, empurramos a lógica de negócio para a view. Os cálculos ocorrem onde os dados residem, no HANA, de forma centralizada, performática e reutilizável por qualquer consumidor.

1. Lógica Condisional com 'CASE'

'Traduzir' códigos técnicos em valores semânticos ou visuais (cores).

```
case overall_status
  when 'A' then 3 -- Positivo (Verde)
  when 'O' then 2 -- Critico (Amarelo)
  when 'X' then 1 -- Negativo (Vermelho)
  else 0          -- Neutro (Cinza)
end as StatusCriticality
```

2. Cálculo de Datas com Funções Nativas

Calcular durações ou datas futuras sem código ABAP.

```
dats_days_between(begin_date, end_date)
as DurationDays
```

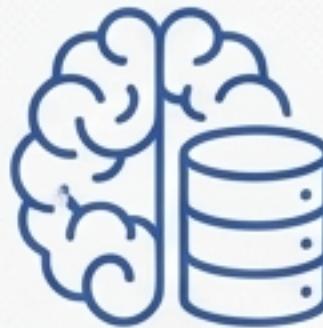
3. Contexto do Utilizador com Variáveis de Sessão

Criar views dinâmicas que se adaptam ao usuário logado.

```
case
  when created_by = $session.user then 'X'
  else ''
end as IsMyTravel
```

O Princípio da Arquitetura Limpa: Separação de Preocupações

Em aplicações reais, uma única view pode ter centenas de campos. Misturar a lógica SQL com dezenas de anotações de UI (`@UI`) torna o código ilegível, rígido e difícil de manter. A solução é separar.



CDS View (.ddls)

Responsabilidad: A Verdade do Dado.

- Estrutura SQL
- Associações
- Cálculos de Negócio (`CASE`)
- Anotações Semânticas (@Semantics)



Metadata Extension (.ddlx)

Responsabilidad: O Estilo do Dado.

- Posição das colunas (@UI.lineItem)
- Filtros de pesquisa (@UI.selectionField)
- Cores e ícones ('criticality')
- Visibilidade dos campos

Analogia Web

CDS View = **HTML** (a estrutura e o conteúdo).

Metadata Extension = **CSS** (o estilo, as cores e o layout).

Mantê-los separados é a melhor prática para um código limpo e flexível.

Passo 6: O Acabamento Final com Metadata Extensions

⚠️ Pré-requisito: Para que a extensão funcione, a CDS View deve conter a anotação de cabeçalho: `@Metadata.allowExtensions: true`.

```
@Metadata.layer: #CORE
annotate view Z_C_TRAVEL with {
    /* Define TravelID como 1ª coluna alta importância, e 1º filtro de pesquisa */
    @UI: { lineItem: [ { position: 10, importance: #HIGH } ],
        selectionField: [ { position: 10 } ] }
    TravelID;

    /* Define AgencyID como 2ª coluna e 2º filtro */
    @UI: { lineItem: [ { position: 20, importance: #HIGH } ],
        selectionField: [ { position: 20 } ] }
    AgencyID;

    /* Define Status como 5ª coluna e vincula a cor ao campo calculado */
    @UI.lineItem: [ { position: 50, importance: #HIGH, criticality: 'StatusCriticality' // Vincula à lógica do CASE }
    ]
    OverallStatus;
}
```

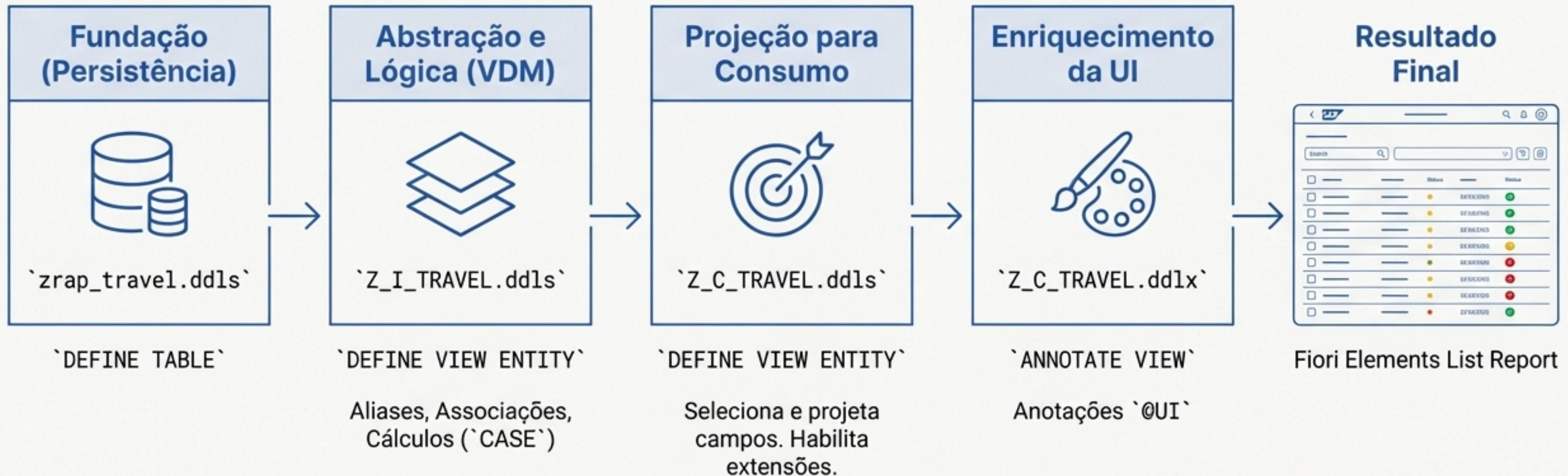
Ordem da Coluna

Responsividade Mobile

Vincula a Lógica ('CASE') à Cor da UI

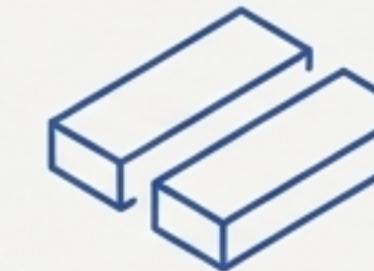
É aqui que a magia acontece. Com anotações, estamos a 'desenhar' a UI. `position` define a ordem, `importance` controla a responsividade em telas móveis e `criticality` conecta a nossa lógica de negócios ('CASE') a cores na tela.

A Construção Completa: A Jornada de um Dado



Cada artefato tem um propósito claro, construindo sobre o anterior para transformar dados brutos numa aplicação de negócio robusta e inteligente.

Os Pilares da Modelagem de Dados Moderna



Code Pushdown

A performance como princípio fundamental. Delegamos o processamento intensivo para o banco de dados HANA, minimizando o tráfego de rede e o uso de memória no servidor de aplicação.

Virtual Data Model (VDM)

A arquitetura em camadas como guia. Abstraímos a complexidade das tabelas físicas, garantindo estabilidade, reutilização e um modelo de dados de negócio claro e semântico.

Separation of Concerns

A manutenção e a flexibilidade como objetivo. Separamos a lógica de dados (.ddls) da lógica de apresentação (.ddlx), resultando em código mais limpo, fácil de manter e adaptável a diferentes UIs.

Com esta planta arquitetural, você está pronto para construir as suas próprias aplicações RAP.