

O Fim do LOOP

Dominando o Code Pushdown em ABAP CDS com Expressões SQL

Uma abordagem moderna para performance e manutenibilidade em S/4HANA.

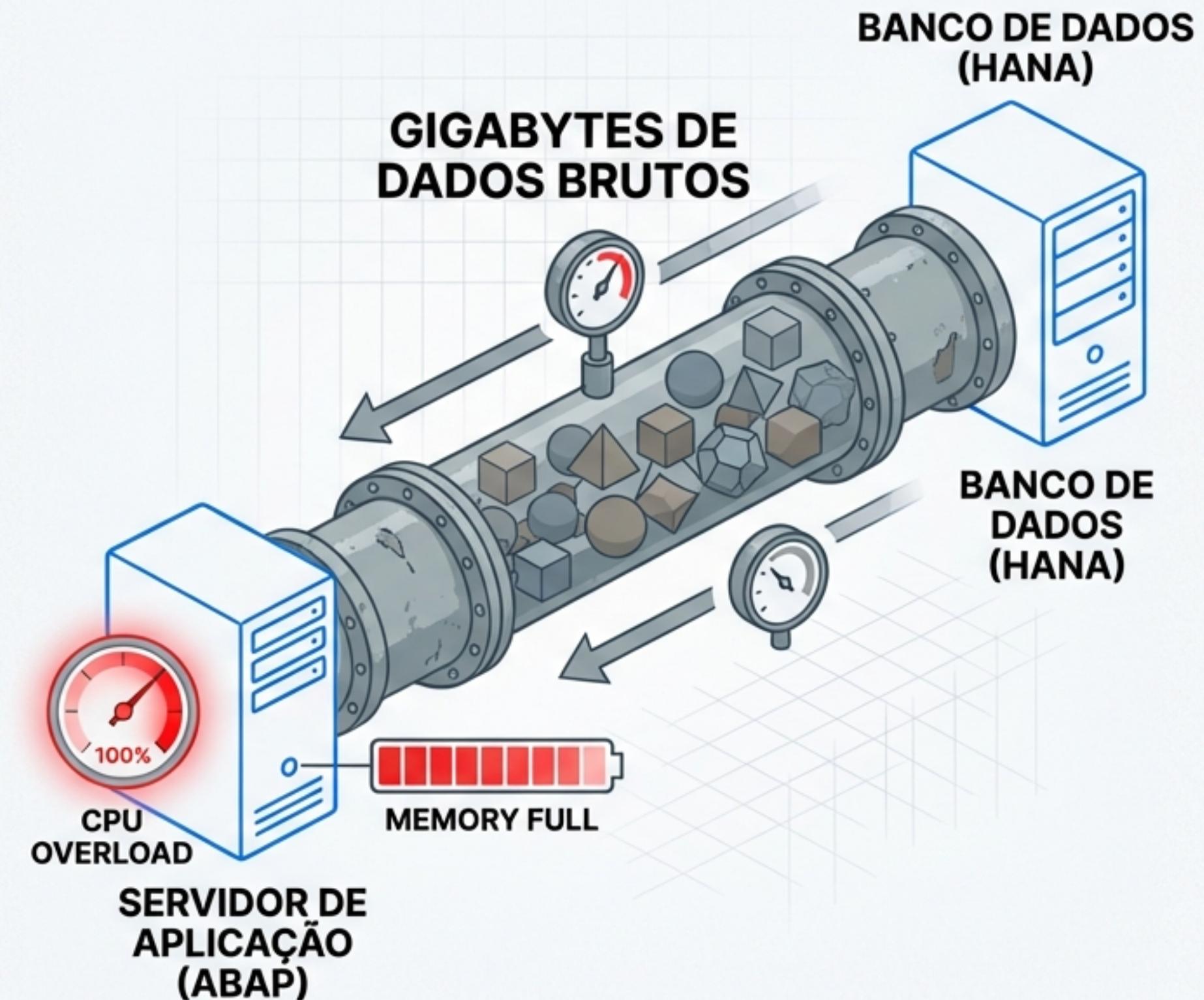
O Paradigma Clássico: “Data-to-Code”

No desenvolvimento ABAP clássico, o padrão era simples e direto:

1. **SELECIONAR** todos os dados brutos do banco de dados.
2. **TRANSFERIR** gigabytes de dados pela rede para o servidor de aplicação.
3. **ITERAR** (LOOP) em tabelas internas para aplicar a lógica de negócio, linha a linha.

O Problema:

Essa abordagem gera um tráfego de rede imenso, consome memória e CPU do servidor de aplicação desnecessariamente e subutiliza o poder do banco de dados moderno (HANA).



A Solução Moderna: “Code-to-Data”

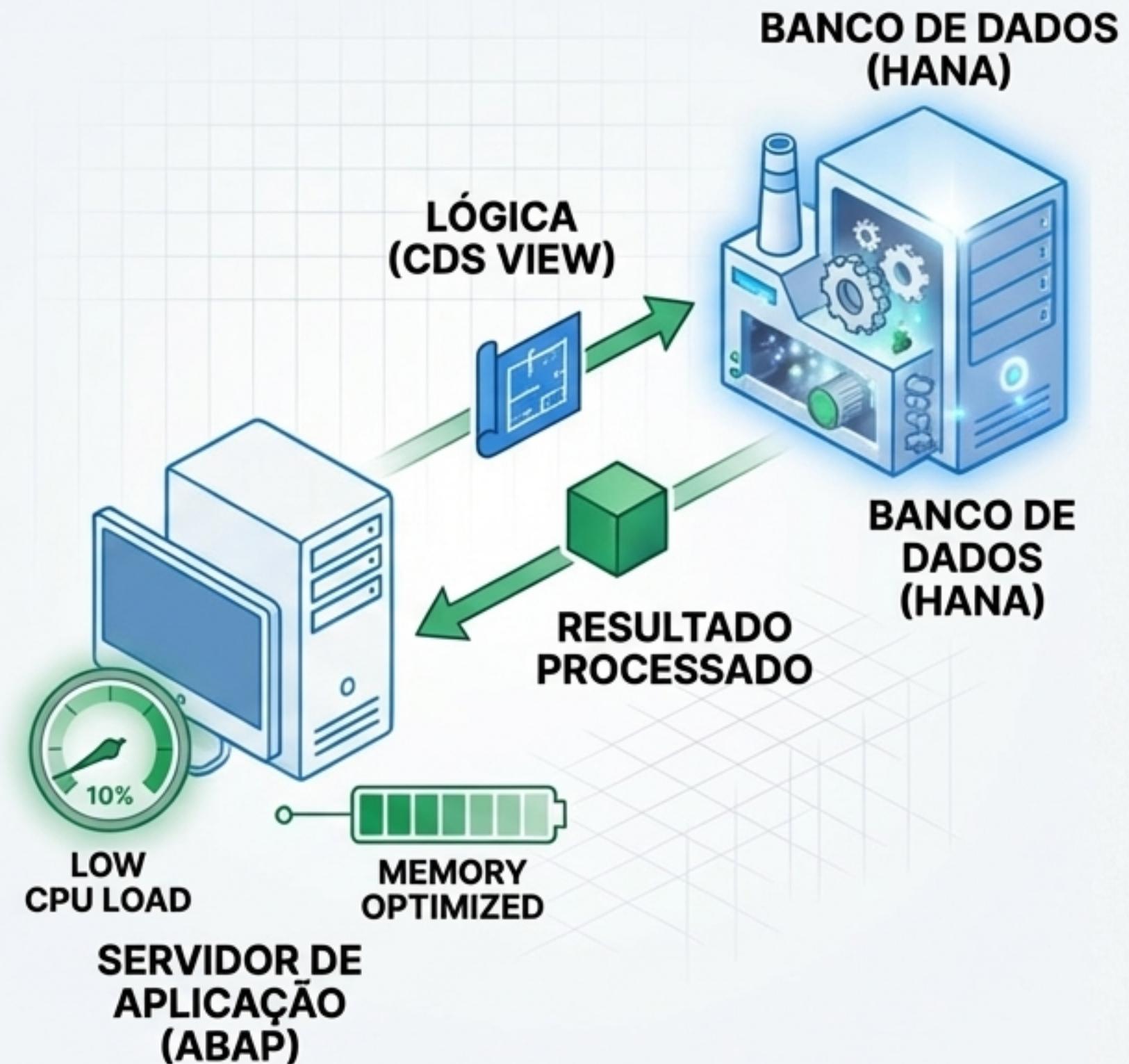
Com o Code Pushdown, invertemos o fluxo. Em vez de trazer os dados até a lógica, nós “empurramos” a lógica para onde os dados residem: o banco de dados.

Como Funciona

A lógica de negócio (cálculos, condições, conversões) é definida diretamente na CDS View. O banco de dados HANA, otimizado para processamento massivo e paralelo, executa essas operações durante a leitura dos dados. O ABAP recebe apenas o resultado final, já processado.

Vantagens:

- ✓ **Performance Radical:** O cálculo ocorre onde os dados estão, eliminando o tráfego de rede e aproveitando o processamento paralelo do HANA.
- ✓ **Manutenção Centralizada:** A regra de negócio fica na View, reutilizável por relatórios, Fiori Apps e APIs, em vez de duplicada em múltiplos programas ABAP.



Ferramenta 1: Lógica Condicional com Expressões CASE

O comando CASE é o equivalente SQL do IF...ELSEIF...ELSE do ABAP. Ele permite criar colunas calculadas cujo valor depende de condições lógicas em outros campos.

****Uso Comum**:** Traduzir códigos técnicos em textos de negócio, diretamente no SELECT.

~~// Antes, em ABAP:~~

```
LOOP AT lt_travel INTO ls_travel.  
  IF ls_travel-overall_status = '0'.  
    ls_travel-status_text = 'Em Aberto'.  
  ELSEIF ls_travel-overall_status = 'A'.  
    ...  
  ENDIF.  
ENDLOOP.
```

// Agora, na CDS View:

```
case overall_status  
  when '0' then 'Em Aberto'  
  when 'A' then 'Aceito'  
  when 'X' then 'Cancelado'  
  else 'Desconhecido'  
end as StatusText
```

“CASE” Avançado: Definindo a “Criticality” para o SAP Fiori

O Conceito

Fiori Elements utiliza um campo numérico de “criticidade” para colorir status e ícones, melhorando a usabilidade. A lógica para definir essa cor agora reside na CDS View.

- 3: Positivo (Verde)**
- 2: Crítico (Amarelo/Laranja)**
- 1: Negativo (Vermelho)**
- 0: Neutro (Cinza)**

	Sales Order	Customer	Status	Date	Value
	2027800	Customer	Aceito	03/08/2023	4.425,00
	2027801	Customer	Atrasado	05/08/2023	4.382,00
	2027013	Customer	Cancelado	03/08/2023	1.403,00

Exemplo de Lógica de Negócio

```
/* Define a cor do status para o Fiori:  
 - Cancelado é sempre Vermelho.  
 - Em Aberto por >30 dias é Amarelo (Atrasado).  
 - Aceito é Verde.  
 */  
case when overall_status = 'X' then 1 -- Vermelho  
    then 1 -- Vermelho (Erro)  
when overall_status = '0' and dats_days_  
    between(created_at, $session.system_date) > 30  
then 2 -- Amarelo (Atrasado)  
when overall_status = 'A' then 3 -- Verde (Sucesso)  
else 0 -- Cinza (Neutro)  
end as StatusCriticality
```

ⓘ Observe a combinação de uma comparação simples (=) com uma função de data (dats_days_between) dentro da mesma condição ‘WHEN’.

Ferramenta 2: Funções Nativas (Aritmética e Datas)

O ABAP CDS oferece uma rica biblioteca de funções que eliminam a necessidade de processamento posterior em ABAP para cálculos comuns.



Funções Aritméticas Essenciais

- **Operadores:** `+`, `-`, `*`, `/` (divisão exata, retorna `FLTP`)
- **Funções:** `div(a, b)` (divisão inteira), `mod(a, b)` (resto), `ceil(x)` (arredondar para cima), `floor(x)` (arredondar para baixo).



Funções de Data Essenciais

- `dats_days_between(data_inicio, data_fim)`: Retorna o número de dias entre duas datas.
- `dats_add_days(data, dias)`: Adiciona ou subtrai dias de uma data.

Exemplo:

```
ceil( total_price * 1.1 ) as PriceWithMargin
```

Exemplo:

```
dats_add_days( begin_date, 30 ) as DueDate
```

Ferramenta 3: Contextualizando com Variáveis de Sessão (\$session)

O Desafio:

Como uma CDS View (um objeto de banco de dados) pode acessar informações do contexto do usuário logado (como seu ID ou a data atual)? Não podemos usar `sy-datum` ou `sy-uname` diretamente no DDL.

A Solução:

O framework provê as **variáveis de sessão**, que são preenchidas pelo banco de dados no momento da execução.

Variáveis Chave

-  **\$session.user**: O ID do usuário logado.
-  **\$session.system_date**: A data atual do servidor.
-  **\$session.system_language**: O idioma de logon do usuário.
-  **\$session.client**: O mandante atual (geralmente implícito).

Exemplo Prático:

```
Criar um flag 'É minha viagem?'  
case  
when created_by = $session.user then 'X'  
else ''  
end as IsMyTravel
```

Ferramenta 4: Garantindo a Integridade com `CAST`

O comando `CAST` é uma função explícita para converter um valor de um tipo de dado para outro. Essencial para manter a segurança de tipos (Type Safety) do SQL e do ABAP.

Quando Usar? Cenários Comuns

-  **Cálculos Aritméticos Precisos:** Converter campos de moeda (`CURR`) ou decimais (`DEC`) para ponto flutuante (`FLTP`) antes de multiplicar ou dividir para evitar perda de precisão.
-  **Concatenação de Strings:** Unir um campo numérico (como um ID) com um texto. O número deve ser convertido para `CHAR` primeiro.
-  **Uniões (`UNION`):** Garantir que as colunas correspondentes em diferentes `SELECTs` de uma `UNION` tenham tipos de dados compatíveis.

Sintaxe e Exemplos

```
// Sintaxe: cast( expressao as TipoAlvo )
```

```
// Convertendo para Float para um cálculo preciso  
cast( total_price as abap.fltp ) as PriceAsFloat
```

```
// Convertendo ID para Char para uso em texto  
cast( travel_id as abap.char(8) ) as TravelIdAsChar
```

Exemplo Prático: Enriquecendo a View de Viagens

Vamos aplicar todos os conceitos em uma única Interface View, **Z_I_TRAVEL_COMPUTED**. O objetivo é criar uma fonte de dados rica para uma aplicação Fiori, pré-calculando todos os campos necessários.

Requisitos de Negócio:

-  1. **Custo Total:** Calcular o preço total somando **total_price** e **booking_fee**.
-  2. **Duração:** Calcular a duração da viagem em dias.
-  3. **Contagem Regressiva:** Mostrar quantos dias faltam para o início da viagem.
-  4. **Status Visual:** Definir a **Criticality** para o Fiori baseada no status.
-  5. **Indicador de Histórico:** Um flag para viagens que já ocorreram.
-  6. **Indicador de Propriedade:** Um flag para viagens criadas pelo usuário logado.

Próximo Slide: Veja o código completo que atende a todos esses requisitos com zero linhas de código ABAP de 'LOOP'. →

O Código Completo: `Z_I_TRAVEL_COMPUTED`

```
@EndUserText.label: 'Viagens com Cálculos Avançados'  
define view entity Z_I_TRAVEL_COMPUTED  
    as select from zrap_travel  
{  
    key travel_uuid,  
        travel_id,  
        begin_date,  
        end_date,  
        total_price,  
        booking_fee,  
        currency_code,  
        overall_status,  
        created_by,  
  
    /* --- 1. Aritmética e CAST --- */  
    @Semantics.amount.currencyCode: 'currency_code'  
    cast( total_price as abap.fltp ) + cast( booking_fee as abap.fltp ) as GrandTotal,  
  
    /* --- 2. Funções de Data --- */  
    dats_days_between( begin_date, end_date ) as DurationDays,  
    dats_days_between( $session.system_date, begin_date ) as DaysUntilStart,  
  
    /* --- 3. Lógica Condisional com CASE --- */  
    case overall_status  
        when 'A' then 3 -- Aceito (Verde)  
        when 'O' then 2 -- Aberto (Amarelo)  
        when 'X' then 1 -- Cancelado (Vermelho)  
        else 0  
    end as StatusCriticality,  
  
    /* --- 4. Variável de Sessão e Lógica --- */  
    case  
        when begin_date < $session.system_date then 'X'  
        else ''  
    end as IsPastTravel,  
    case  
        when created_by = $session.user then 'X'  
        else ''  
    end as IsCreatedByMe  
}
```

O Próximo Nível: Views Analíticas com Agregações

Além do 1:1

Até agora, vimos views que retornam uma linha para cada registro da tabela base. Mas o CDS é igualmente poderoso para criar views que sumarizam dados (OLAP).

Ferramentas de Agregação

- **Funções:** `SUM()`, `COUNT()`, `AVG()`, `MIN()`, `MAX()`
- **Cláusula Obrigatória:** `GROUP BY`

A Regra de Ouro do `GROUP BY`

Se você usar uma função de agregação (como `SUM`) em qualquer campo, **todos** os outros campos na lista de seleção que não são agregados devem obrigatoriamente aparecer na cláusula `GROUP BY`.

Exemplo: Contar viagens e somar gastos por cliente.

```
define view entity Z_I_TRAVEL_STATS
  as select from zrap_travel
{
  key customer_id,
  count(*)           as NumberOfTravels, --
  sum( total_price ) as TotalSpent,
  avg( total_price as abap.dec(15,2) ) as
  AverageTicketPrice, --
  currency_code      as CurrencyCode
}
group by
  customer_id,
  currency_code --
```

O Veredito: Comparativo Direto entre Paradigmas

Tarefa	Abordagem Clássica (Data-to-Code)	Abordagem CDS (Code-to-Data)
Traduzir Status	LOOP na tabela interna com IF/ELSEIF.	<input checked="" type="checkbox"/> Uma coluna com CASE na CDS View.
Calcular Duração	LOOP e subtrair datas em variáveis ABAP.	<input checked="" type="checkbox"/> Uso da função dats_days_between() na View.
Totalizar por Grupo	LOOP com COLLECT ou AT END OF.	<input checked="" type="checkbox"/> Uso de SUM() com a cláusula GROUP BY na View.
Usar Data Atual	Acessar a variável sy-datum no programa.	<input checked="" type="checkbox"/> Usar a variável \$session.system_date na View.
Filtrar por Usuário	SELECT ... WHERE user = sy-uname.	<input checked="" type="checkbox"/> WHERE created_by = \$session.user na View.

Glossário Essencial

Code Pushdown

A prática de mover a lógica de processamento de dados do servidor de aplicação para o banco de dados.

Expressão SQL (Expression)

Qualquer comando que gera um novo valor em tempo de execução, como um cálculo (`total * 1.1`), uma condição (`CASE...`), ou uma função (`date_add_days(...)`).

CAST

Função SQL para converter explicitamente um valor de um tipo de dado para outro (ex: NUMC para INT4). Essencial para a segurança de tipos em cálculos e uniões.

Variável de Sessão (\$session)

Variáveis de contexto (`$session.user`, `$session.system_date`) que o banco de dados provê em tempo de execução, permitindo que a View reaja ao ambiente do usuário.

Agregação (Aggregation)

Funções (`SUM`, `COUNT`, `AVG`) que condensam múltiplos registros em um único resultado de resumo. Exigem o uso da cláusula `GROUP BY`.

Criticidade (Criticality)

Um conceito da UI do SAP Fiori onde valores numéricos (0-3) são mapeados para cores semânticas (Cinza, Vermelho, Amarelo, Verde) para indicar o status de um item de forma visual.

Teste seu Conhecimento

Q1: Por que usamos \$session.system_date em uma CDS View em vez de sy-datum?

R: Porque a CDS View é um objeto do banco de dados (DDL), enquanto sy-datum é uma variável de memória do servidor de aplicação ABAP. O banco não tem acesso direto à memória do ABAP. \$session.system_date é o valor de contexto correto no nível do banco de dados.

Q2: Para transformar o status 'A' no valor numérico 3 para o Fiori, qual expressão usamos e qual o benefício?

R: Usamos a expressão CASE. O benefício é que a lógica de apresentação ('A' significa Verde/3) fica centralizada na View, e pode ser reutilizada por qualquer App Fiori sem que o front-end precise conhecer a regra de negócio.

Q3: O que acontece se eu usar SUM(price) em uma View sem a cláusula GROUP BY?

R: Ocorrerá um erro de sintaxe. Funções de agregação como SUM precisam saber como agrupar as linhas. A cláusula GROUP BY especifica quais campos definem esses grupos (ex: GROUP BY customer_id).