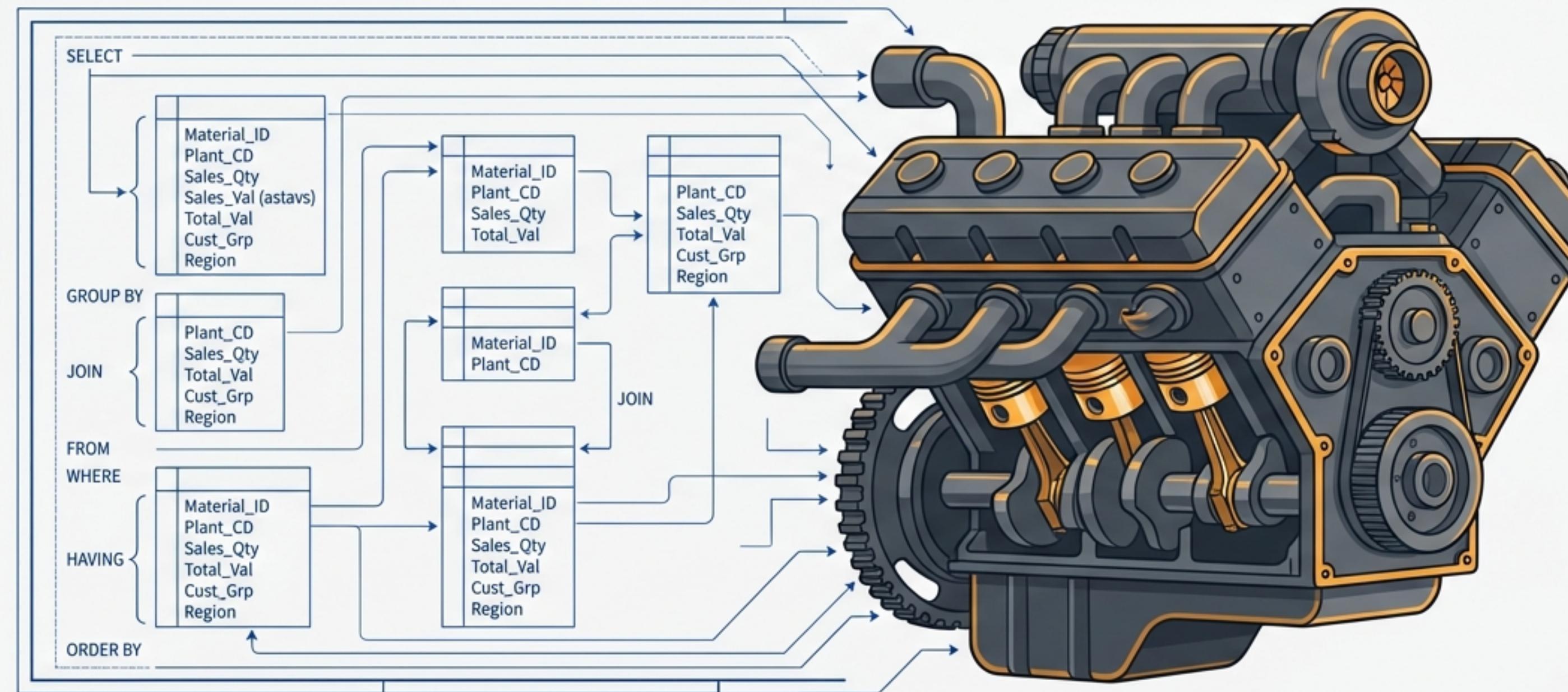


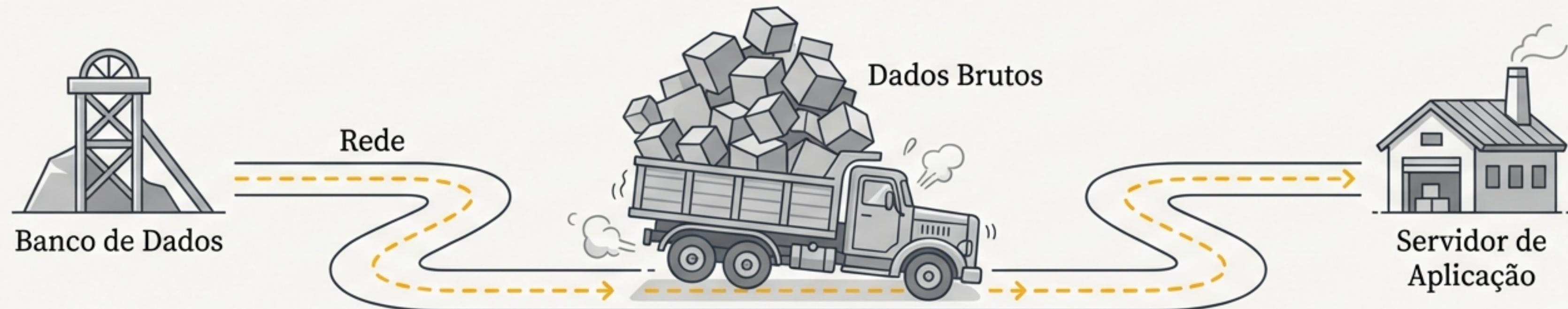
A Revolução Silenciosa no ABAP: Dominando o Code Pushdown para Performance Máxima

De 'trazer os dados ao código' para 'enviar o código aos dados' no SAP HANA.



O Inimigo da Performance: O Paradigma Clássico

Por anos, a lógica foi: "selecionar tudo (`SELECT *`), jogar em uma tabela interna e processar com `LOOP`". Em bancos de dados modernos como o SAP HANA, esse padrão cria um gargalo massivo de performance. O custo de mover milhões de linhas pela rede apenas para processá-las no servidor de aplicação é proibitivo.



" O JEITO ANTIGO (LENTO)

```
SELECT * FROM zrap_travel INTO TABLE @lt_travel.
```

```
LOOP AT lt_travel ASSIGNING FIELD-SYMBOL(<ls_travel>).
```

" Cálculos, lógicas IF/ELSE, agregações...

```
<ls_travel>-grand_total = <ls_travel>-total_price + <ls_travel>-booking_fee.
```

```
IF <ls_travel>-total_price > 1000.
```

```
..."
```

```
ENDIF.
```

```
ENDLOOP.
```

Evitar

A Mudança de Paradigma: Enviando o Código até os Dados

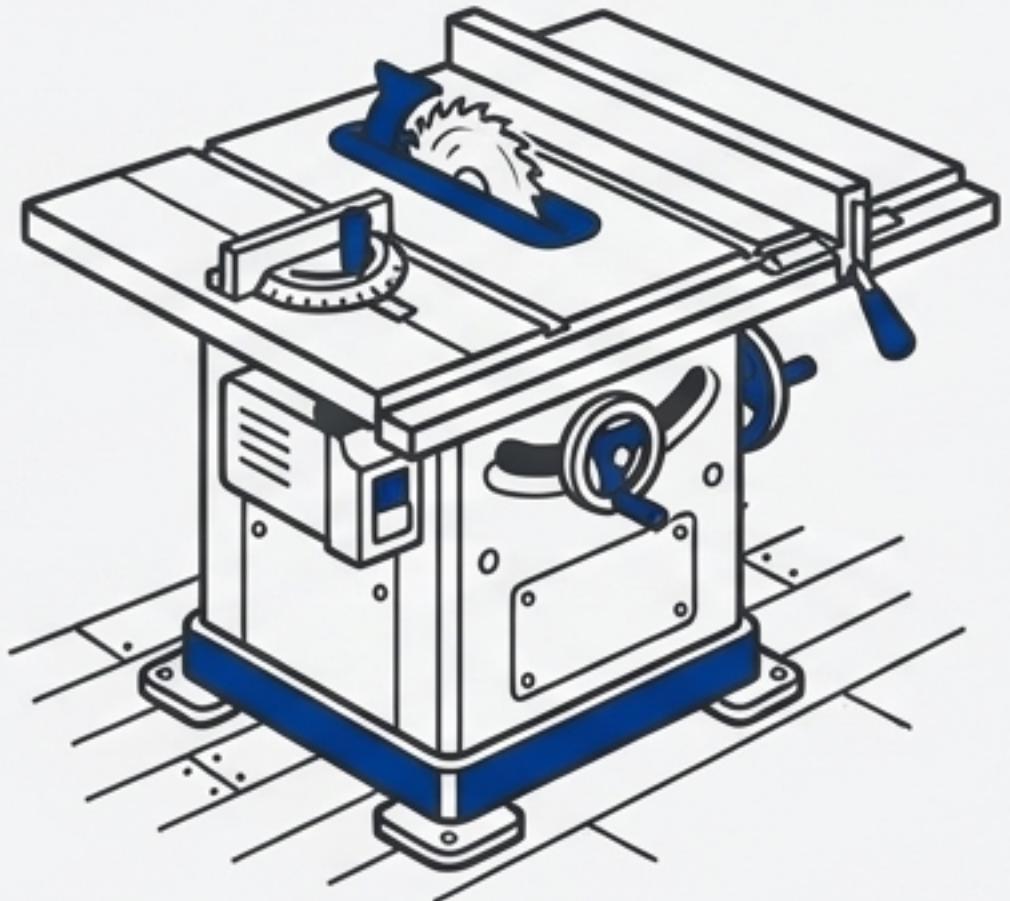
A solução é inverter a lógica. Em vez de trazer dados massivos para o nosso código, nós enviamos o código (cálculos, lógicas e agregações) para ser executado diretamente no banco de dados. O banco de dados é otimizado para operar em grandes conjuntos de dados, retornando para a aplicação apenas o resultado final, já processado e agregado.



Menos tráfego de rede. Processamento mais rápido. Código mais limpo.

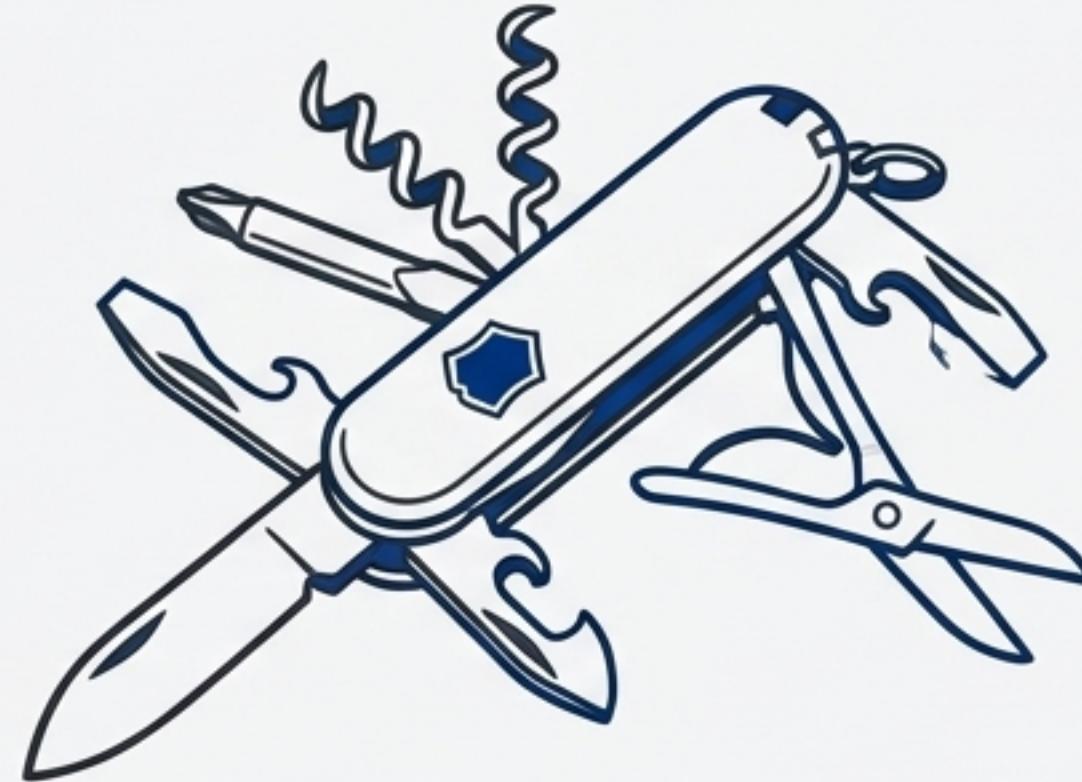
A Ferramenta Certa para o Trabalho: ABAP SQL vs. CDS Views

A escolha depende da **reutilização**. Ambas as tecnologias aplicam o Code Pushdown, mas servem a propósitos diferentes.



CDS View

Ideal para definir modelos de dados reutilizáveis. Se a lógica (ex: cálculo de imposto) for usada em múltiplos relatórios e apps, ela pertence a uma CDS View. É a sua "Single Source of Truth".



ABAP SQL Moderno

Perfeito para lógica ad-hoc, específica para um método ou relatório. Se a query é única para um processamento batch, escreva o SQL diretamente no código ABAP. Não polua o dicionário com views de uso único.

O Toolkit do Code Pushdown: Expressões Aritméticas

Lado Aplicação (O Jeito Clássico)

```
DATA: ls_result LIKE LINE OF lt_results.  
  
LOOP AT lt_travel INTO DATA(ls_travel).  
    ls_result-travel_id = ls_travel-travel_id.  
    " Cálculo manual no ABAP  
    ls_result-grand_total = ls_travel-total_price +  
    ls_travel-booking_fee.  
    APPEND ls_result TO lt_results.  
ENDLOOP.
```

Lado Banco de Dados (Code Pushdown)

```
" A matemática acontece diretamente no banco!  
SELECT FROM zrap_travel  
FIELDS travel_id,  
       ( total_price + booking_fee ) AS grand_total,  
       " Use CAST para garantir o tipo de dado correto  
       CAST( total_price * '0.9' AS CURR( 15, 2 ) ) AS  
       discounted_price,  
       " Divisão com precisão decimal  
       DIVISION( total_price, 100, 2 ) AS price_index  
INTO TABLE @DATA(lt_results).
```

Cuidado com Nulos!

Em SQL, `5 + NULL = NULL`. Para evitar que um campo nulo anule seu cálculo, use a função `COALESCE`.

`(total_price + COALESCE(booking_fee, 0)) AS grand_total`

O Toolkit do Code Pushdown: Manipulação de Strings

Lado Aplicação (O Jeito Clássico)

```
LOOP AT lt_customers INTO DATA(ls_customer).
  " Concatenação e formatação no ABAP
  CONCATENATE ls_customer-first_name ls_customer-last_name
    INTO ls_customer-full_name SEPARATED BY space.
  ls_customer-upper_name = to_upper( ls_customer-last_name ).
  APPEND ls_customer TO lt_formatted_customers.
ENDLOOP.
```

Lado Banco de Dados (Code Pushdown)

```
SELECT FROM /dmo/customer
FIELDS customer_id,
  " Concatenação simples com operador &&
  first_name && ' ' && last_name AS raw_name,
  " Função dedicada com separador (mais limpa)
  concat_with_space( first_name, last_name, 1 ) AS full_name,
  " Normalização para maiúsculas
  upper( last_name ) AS upper_name,
  " Extração de parte do texto (substring)
  substring( last_name, 1, 3 ) AS short_name
INTO TABLE @DATA(lt_names).
```

O Toolkit do Code Pushdown: Lógica Condisional com `CASE`

Substitua blocos `IF/ELSE` ou `CASE` dentro de loops por uma expressão `CASE` declarativa diretamente no seu `SELECT`.

Lado Aplicação (O Jeito Clássico)

```
LOOP AT lt_travel ASSIGNING FIELD-SYMBOL(<travel>).
CASE <travel>-overall_status.
    WHEN 'A'. <travel>-status_text = 'Aceito'.
    WHEN 'X'. <travel>-status_text = 'Rejeitado'.
    ELSE.    <travel>-status_text = 'Pendente'.
ENDCASE.
ENDLOOP.
```

Lado Banco de Dados (Code Pushdown)

```
SELECT FROM zrap_travel
FIELDS travel_id,
    " Simple CASE: Tradução de Status 1-para-1
CASE overall_status
    WHEN 'A' THEN 'Aceito'
    WHEN 'X' THEN 'Rejeitado'
    ELSE 'Pendente'
END AS status_text,

    " Complex CASE: Categorização baseada em condições
CASE
    WHEN total_price < 1000 THEN 'Econômica'
    WHEN total_price BETWEEN 1000 AND 5000 THEN 'Executiva'
    ELSE 'Primeira Classe'
END AS price_category
INTO TABLE @DATA(lt_status).
```

O Coração da Análise: Agregação de Dados

Para relatórios de totais, médias ou contagens, o banco de dados é ordens de magnitude mais rápido. Usar `LOOP` ou `COLLECT` no ABAP para isso é um antipadrão de performance. Encontrado segundo de JetBrains Mono.

Funções de Agregação Essenciais

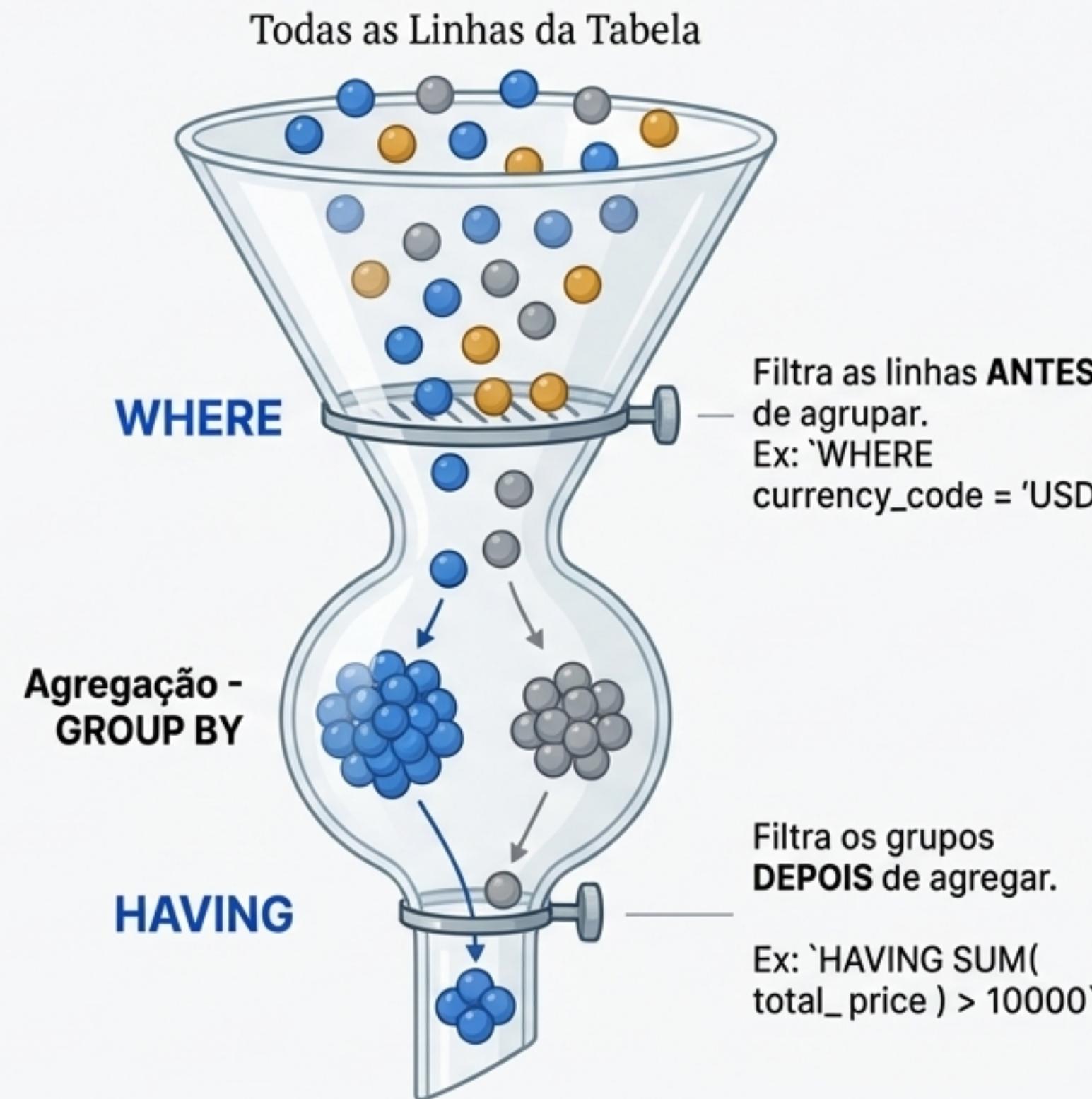
- `SUM(campo)`: Soma os valores de um campo.
- `COUNT(*)` ou `COUNT(campo)`: Conta o número de linhas.
- `AVG(campo)`: Calcula a média.
- `MIN(campo) / MAX(campo)`: Encontra o valor mínimo/máximo.

A Regra Fundamental

Ao usar uma função de agregação (`SUM`, `COUNT`, etc.), qualquer outro campo na lista `FIELDS` que não está sendo agregado deve obrigatoriamente estar na cláusula `GROUP BY`.

O banco precisa saber ‘somar o quê, agrupado por quem?’.

Filtrando Dados: A Diferença Crucial entre 'WHERE' e 'HAVING'



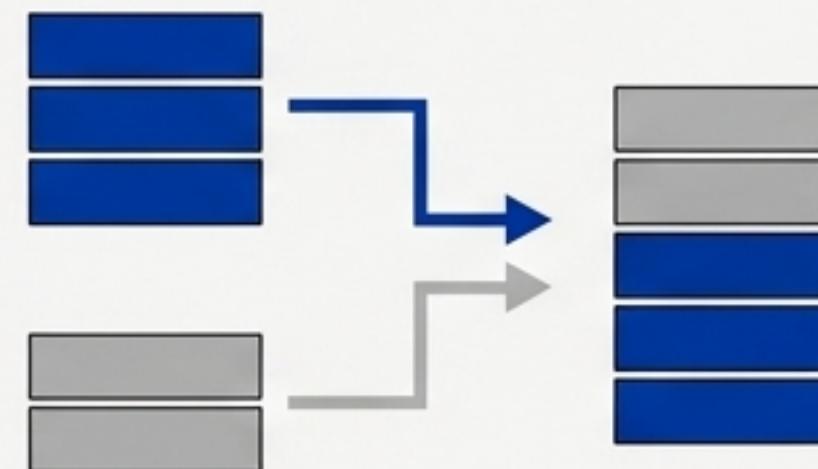
```
SELECT customer_id,  
       SUM( total_price ) AS total_spent  
  FROM zrap_travel  
  " 1. Filtra as linhas ANTES de somar  
 WHERE begin_date >= '20230101'  
  GROUP BY customer_id  
  " 2. Filtra os grupos DEPOIS de somar  
 HAVING SUM( total_price ) > 50000  
 INTO TABLE @DATA(lt_analytics).
```

O Toolkit do Code Pushdown: Combinando Resultados com 'UNION'

Use `UNION` para juntar resultados de múltiplas queries em uma única ida ao banco de dados. O requisito é que o número e os tipos de colunas sejam compatíveis.

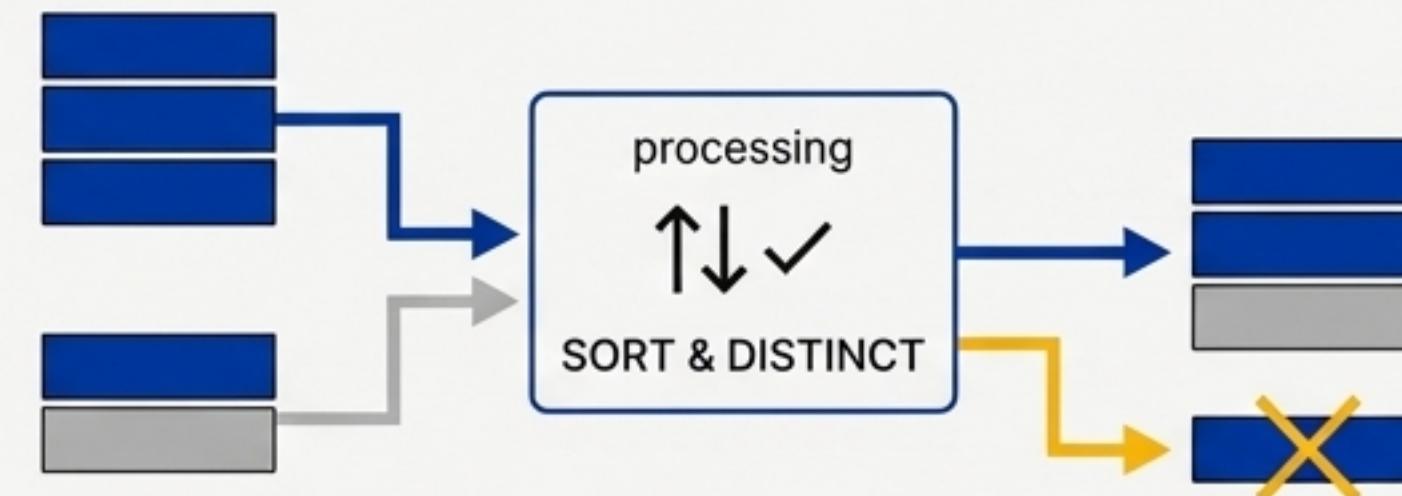
'UNION ALL' (Rápido)

Simplesmente anexa os resultados do segundo `SELECT` ao primeiro. Não verifica duplicatas. É a opção mais performática e ideal na maioria dos casos.



'UNION' (Lento)

Junta os resultados E executa um passo extra para ordenar e remover quaisquer linhas duplicadas entre os dois conjuntos. Use apenas se a unicidade for absolutamente necessária.



```
" Seleciona Voos Ativos
SELECT FROM /dmo/connection
FIELDS carrier_id, connection_id, distance
WHERE distance > 2000
UNION ALL " A opção mais rápida
" Seleciona Voos de uma tabela de histórico
SELECT FROM /dmo/conn_hist
FIELDS carrier_id, connection_id, distance
WHERE distance > 2000
INTO TABLE @DATA(lt_all_long_flights).
```

A Sinfonia do SQL: Um Exemplo Analítico Completo

Vamos construir um relatório de performance de agências, combinando agregação, cálculo, lógica condicional e filtros pós-agregação em uma única e poderosa query.

```
SELECT FROM zrap_travel
FIELDS agency_id,
      currency_code,
      COUNT( * ) AS number_of_travels,    // <--- 1
      SUM( total_price ) AS total_amount, // <--- 1
      AVG( total_price AS DEC( 15, 2 ) ) AS average_ticket, // <--- 2
      CASE
          WHEN SUM( total_price ) > 100000 THEN 'Platinum Partner'
          WHEN SUM( total_price ) > 10000  THEN 'Gold Partner'
          ELSE 'Standard Partner'
      END AS partner_category // <--- 3
GROUP BY agency_id, currency_code
HAVING  SUM( total_price ) > 1000 // <--- 4
ORDER BY total_amount DESCENDING
INTO TABLE @DATA(lt_report).
```

[1] AGREGAÇÃO
Contagem e Soma

[2] EXPRESSÃO ARITMÉTICA
Média com casting

[3] LÓGICA CONDICIONAL
Classificação com CASE

[4] FILTRO PÓS-AGREGAÇÃO
Foco em parceiros relevantes

Guia de Campo: ABAP Clássico vs. Code Pushdown

Cenário	ABAP Clássico (Evitar)	ABAP SQL (Recomendado)	Vantagem SQL
Soma de Totais	`LOOP`, acumular em variável.	`SELECT SUM(...)`	Menor tráfego de rede, uso de índices de coluna.
Tradução de Status	`LOOP`, `IF/ELSE`, modificar tabela.	`SELECT CASE ... END AS ...`	Lógica centralizada, retorno já formatado.
Unir Resultados	Dois `SELECT`s, `LOOP` e `APPEND`.	`SELECT ... UNION ALL ...`	Uma única ida ao banco (Roundtrip).
Filtrar por Total	`LOOP`, calcular, `DELETE` se $< X$.	`SELECT ... HAVING SUM(...) > X`	O banco de dados só retorna o que realmente interessa.

Guia de Campo: Glossário Essencial

Função de Agregação

Funções SQL (`SUM`, `AVG`, `COUNT`) que operam em um conjunto de linhas para retornar um único valor resumido.

GROUP BY

Cláusula obrigatória ao misturar campos normais e funções de agregação. Define como os dados serão agrupados para o cálculo.

HAVING

Cláusula que filtra os resultados após a agregação. Usada para condições sobre valores summarizados (ex: HAVING SUM(...) > 100).

WHERE

Cláusula que filtra as linhas antes da agregação. Usada para condições sobre dados brutos.

UNION ALL / UNION

Operadores que combinam resultados de múltiplos SELECT's. UNION ALL é rápido e mantém todas as linhas; UNION remove duplicatas e é mais lento.

COALESCE

Função que retorna o primeiro valor não nulo de uma lista. Indispensável para evitar que valores NULL anulem cálculos aritméticos.

O seu código ABAP nunca mais será o mesmo.

Você agora tem as ferramentas para construir aplicações mais rápidas, limpas e eficientes, movendo a lógica para onde ela pertence: junto aos dados.

