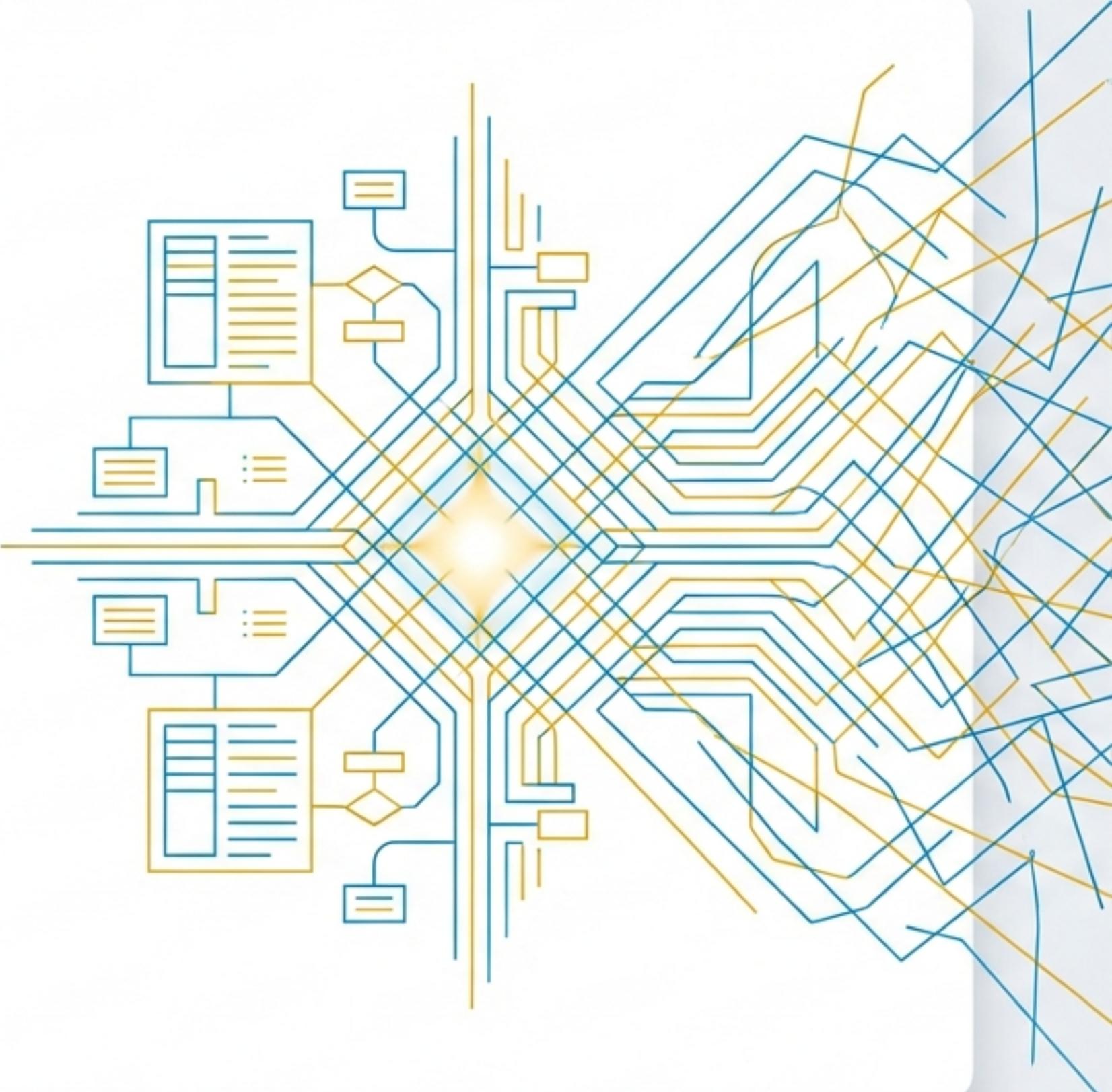


ABAP Moderno: **Dominando Tabelas** **Internas Complexas**

Estratégias de Performance, Sintaxe
Moderna e Otimização com RAP



Objetivos de Aprendizagem

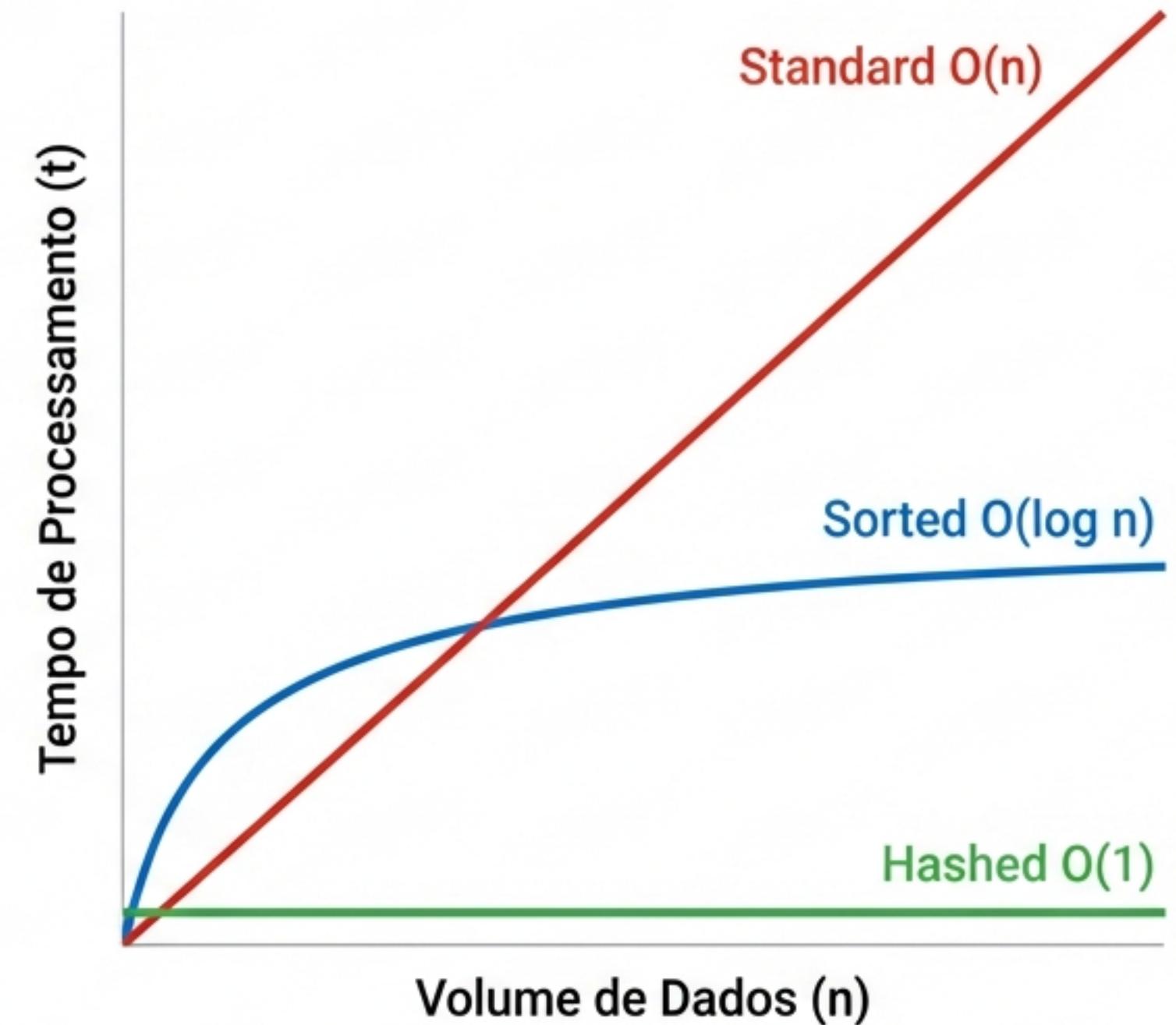
- ✓ **Distinguir tecnicamente** entre tabelas Standard, Sorted e Hashed, compreendendo a complexidade algorítmica (Big O Notation).
- ✓ **Utilizar Expressões de Tabela** (`lt_tab[...]`) para leituras diretas, encadeamentos e gerenciamento de exceções.
- ✓ **Manipular dados** com agilidade usando o operador construtor **VALUE** e a adição de linhas com **BASE**.
- ✓ **Implementar Chaves Secundárias** para otimizar buscas em tabelas sem alterar a estrutura primária.

O Impacto da Escolha: Complexidade Algorítmica

No S/4HANA, processar milhões de registros exige a estrutura correta. A escolha errada resulta em **Timeouts**.

Big O Notation

- ✓ **$O(n)$ - Linear:** O tempo dobra se o volume de dados dobrar.
- ✓ **$O(\log n)$ - Logarítmico:** Extremamente rápido, divide a busca ao meio.
- ✓ **$O(1)$ - Constante:** Tempo de resposta imediato, independente do volume.



Standard vs. Sorted: Flexibilidade ou Ordem?

Standard Table (O Coringa)

- **Estrutura:** Array lógico. Ordem de inserção (APPEND).
- **Busca:** Linear $O(n)$. Percorre linha a linha.
- **Uso Ideal:** Listas pequenas (< 100 linhas) ou buffers de inserção sequencial.

Sorted Table (A Árvore)

- **Estrutura:** Mantém dados ordenados automaticamente.
- **Busca:** Binária $O(\log n)$. Divide a tabela para encontrar o valor.
- **Trade-off:** Inserção (INSERT) é mais lenta que na Standard, pois o sistema precisa calcular a posição correta.

Hashed Tables: Velocidade Máxima



Acesso Direto via Algoritmo de Hash.

Performance: $O(1)$ (Tempo Constante)

O tempo de resposta é o mesmo para 10 ou 10 milhões de linhas.

Regra de Ouro

Exige **UNIQUE KEY**. Não permite chaves duplicadas.

Restrição

Não acessível por índice numérico (sy-tabix), apenas pela chave.

Uso Ideal

Caches e Tabelas Mestras (Clientes, Materiais) onde o acesso é sempre pelo ID.

Modernizando a Leitura: Expressões de Tabela

Legado (Read Table)

```
READ TABLE lt_flights INTO ls_flight  
    WITH KEY carrier_id = 'LH'.  
IF sy-subrc = 0.  
    " Processar  
ENDIF.
```

Moderno (Table Expression)

```
DATA(ls_flight) = lt_flights[  
    carrier_id = 'LH' ].
```



Sintaxe funcional (`itab[...]`) que aproxima o ABAP de linguagens como Java ou Python. Adeus `sy-subrc`.

Tratamento de Exceções e Encadeamento

Exceções (Exceptions)

Diferente do `sy-subrc = 4`, a falha na expressão lança uma exceção.

```
TRY.  
    DATA(row) = lt_tab[ key = val ].  
CATCH cx_sy_itab_line_not_found.  
    " Tratar erro (Opcional)  
ENDTRY.
```

Encadeamento (Chaining)

Acesso direto a campos ou sub-tabelas sem *Work Area*.

```
" Acessa o preço diretamente  
DATA(lv_price) = lt_products[ id =  
'123' ]-price.
```

Funções Predicativas: Verificações Inteligentes



****line_exists(...)****

- Substitui `READ ... TRANSPORTING NO FIELDS`.
- Retorna um booleano. Ideal para `IF`.

```
IF line_exists( lt_tab[ id = 'X' ] ).
```

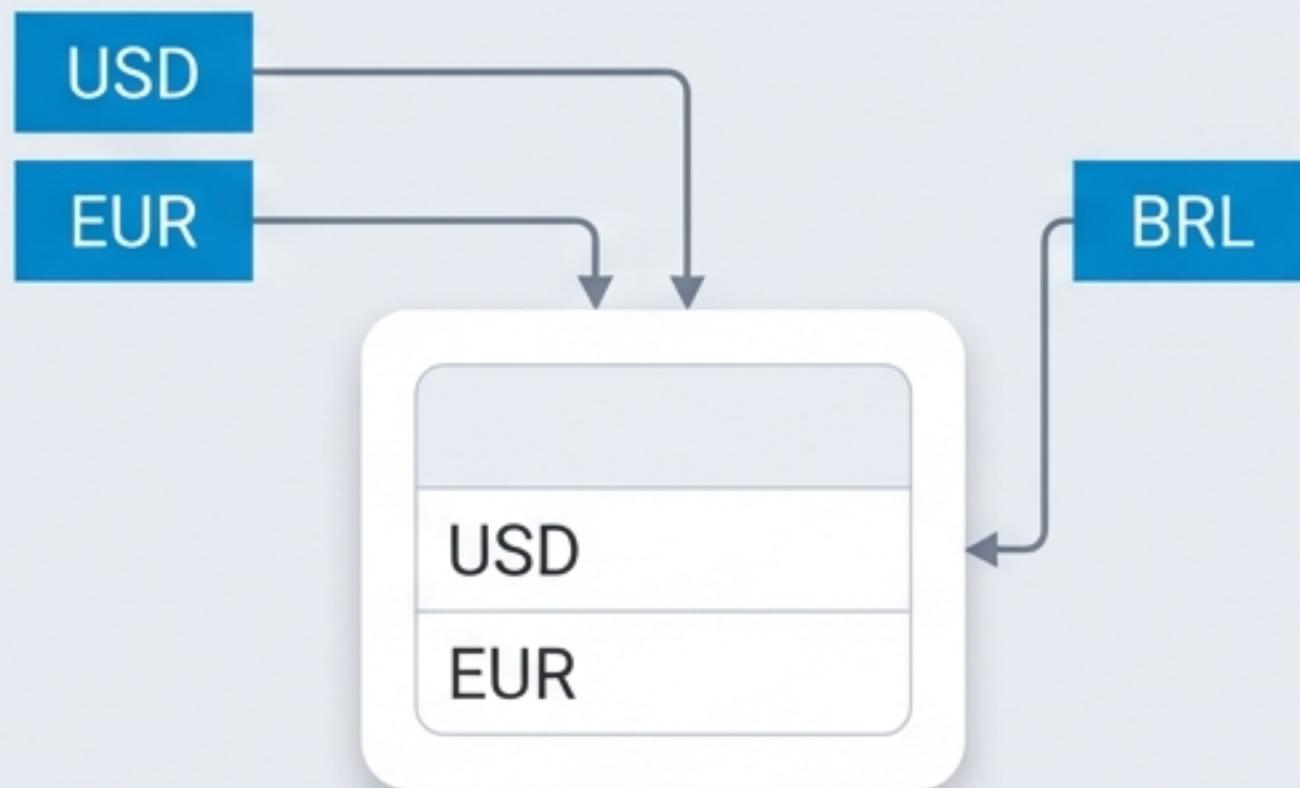


****line_index(...)****

- Retorna o número da linha (`sy-tabix`) ou 0 se não encontrar.
- Acesso direto ao metadado da tabela.

```
DATA(idx) = line_index( lt_tab[ key = 'X' ] ).
```

O Operador VALUE: Construção e Carga



Inicialização

Criação em massa limpa e visual.

```
DATA(lt_curr) =  
  VALUE ty_tab(  
    ( code = 'USD' )  
    ( code = 'EUR' )  
  ).
```

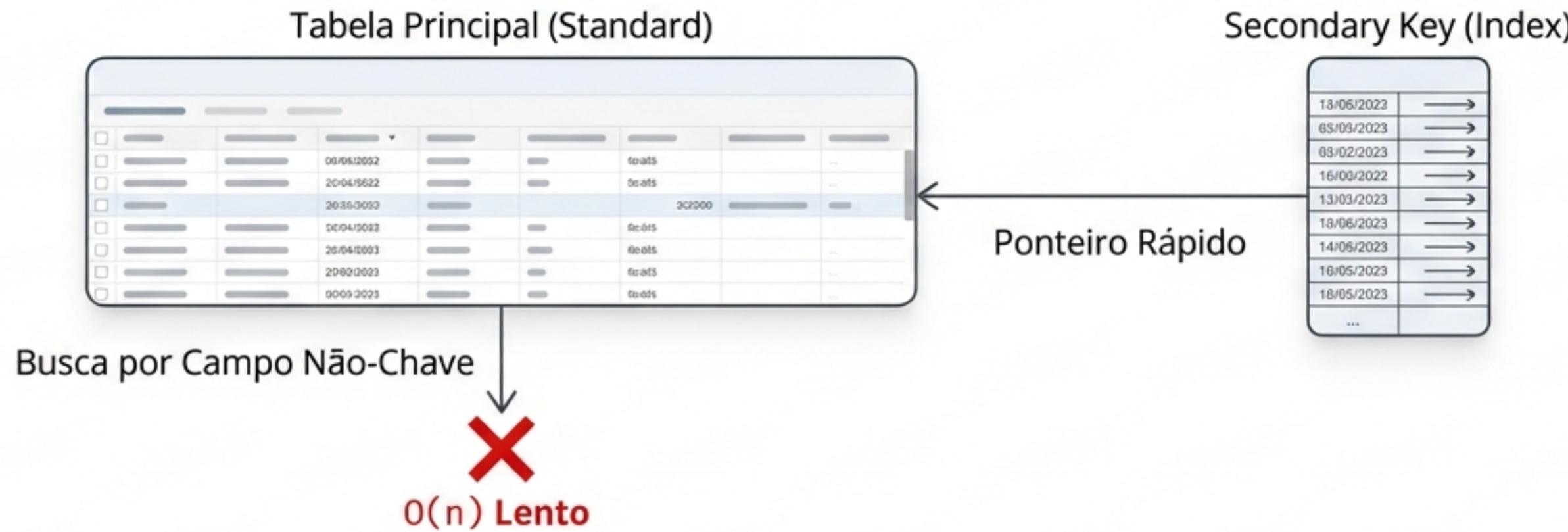
O Poder do BASE

Sem `BASE`, a tabela é limpa antes da inserção.

Com `BASE`, anexa novas linhas preservando o conteúdo existente.

```
lt_curr = VALUE #(  
  BASE lt_curr  
  ( code = 'BRL' )  
). 
```

Chaves Secundárias: Otimização Cirúrgica



- **O Problema:** Filtrar uma tabela Standard gigante por um campo que *não* é a chave primária (ex: Data de Venda). Isso gera uma busca **O(n)** lenta.
- **A Solução:** Criar uma SECONDARY KEY (Sorted ou Hashed) na definição da tabela.
- **O Resultado:** Transforma a busca linear em **O(log n)** ou **O(1)** sem alterar a estrutura base.

Estudo de Caso: Performance na Prática

```
" Definição da Tabela com Chave Secundária  
TYPES ty_prod_hashed_table TYPE HASHED TABLE OF ty_product  
    WITH UNIQUE KEY id  
    WITH NON-UNIQUE SORTED KEY sk_cat COMPONENTS category.
```

...

```
" Ação: Busca por 'Categoria' (não é a chave primária)  
" O uso de USING KEY ativa a busca binária O(log n)  
LOOP AT lt_products INTO DATA(ls_prod) USING KEY sk_cat  
    WHERE category = 'HW'.
```

```
    out->write( |Produto: { ls_prod-name }| ).  
ENDLOOP.
```

Cenário: Tabela Hashed por ID, mas busca por Categoria.

Sem **USING KEY**, esta operação forçaria um loop **linear** lento em uma tabela Hashed.

Pontos de Atenção: Legado vs. Moderno

Operação	Legado (Antigo)	Moderno (Novo)
Ler Linha	READ TABLE ... INTO wa	wa = tab[...]
Verificar	TRANSPORTING NO FIELDS	line_exists(...)
Obter Índice	Verificar sy-tabix	line_index(...)
Adicionar	APPEND	VALUE ... BASE
Performance	LOOP Simples	LOOP USING KEY

Código limpo e performático é o novo padrão.

Glossário Técnico

Standard Table

Acesso não otimizado (Linear $O(n)$). Ideal para índices/buffers.

Hashed Table

Acesso otimizado por Hash (Constante $O(1)$). Chave única obrigatória.

Sorted Table

Mantida ordenada (Binária $O(\log n)$). Boa para leitura, lenta na inserção.

Secondary Key

Índice adicional para buscas rápidas em campos não-chave.

Big O Notation

Medida de eficiência ($O(1) > O(\log n) > O(n)$).

Table Expression

Sintaxe `itab[...]`. Leitura direta funcional.

Quiz de Fixação

Q1

Qual tabela garante acesso $O(1)$, mas exige chave única?

Q2

Qual exceção ocorre se `itab[...]` falhar na busca?

Q3

Como otimizar filtro por campo não-chave em tabela Standard?

Q4

Qual o "trade-off" (custo) da Tabela Sorted no INSERT?

Código limpo não é apenas estética. É performance e escalabilidade.

Revise seus códigos antigos. Substitua READ TABLE por Expressões Modernas
e implemente Chaves Secundárias onde a performance for crítica.