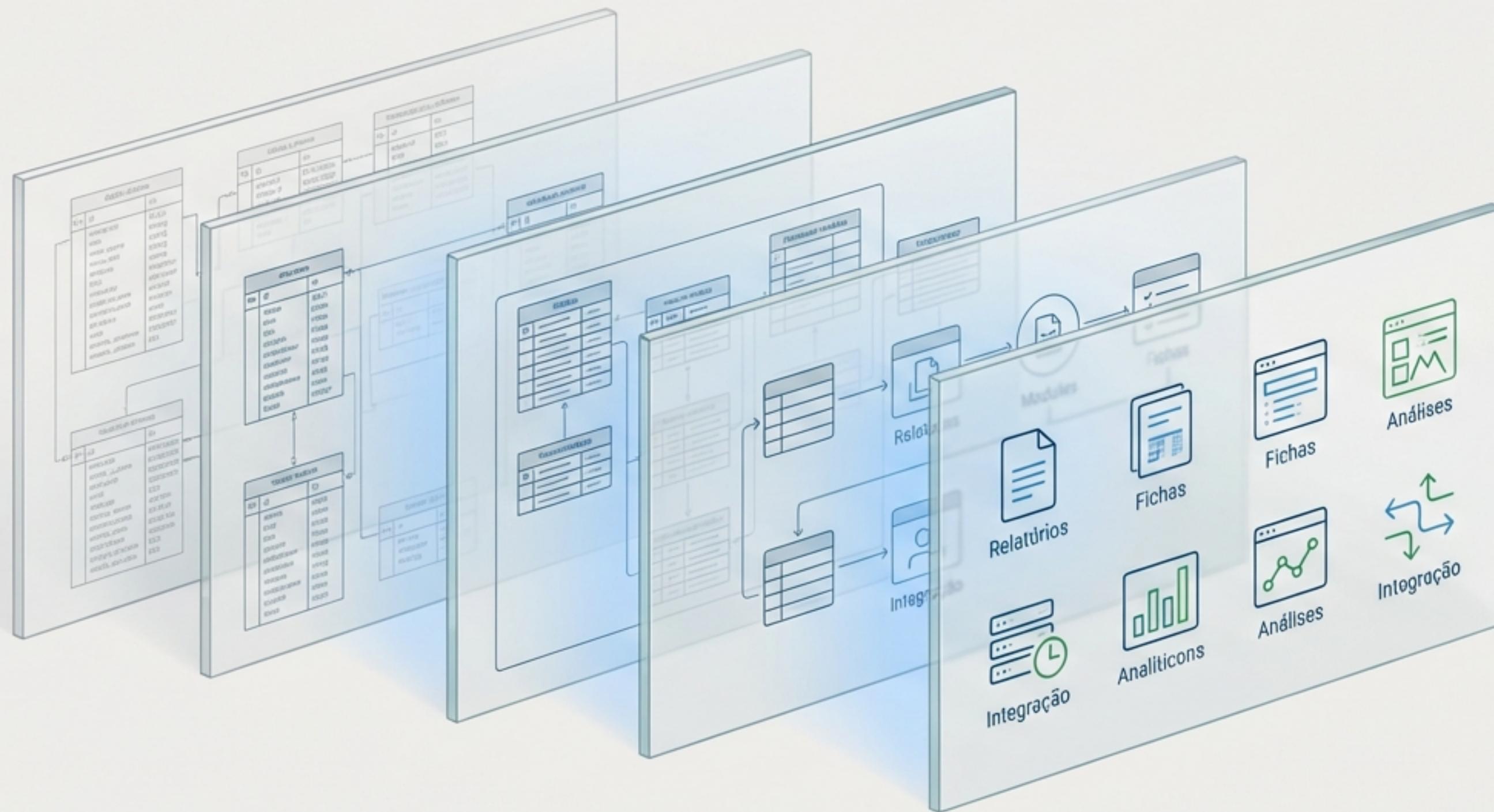


# Dominando Estruturas de Dados em ABAP

# A base da transformação de dados no modelo RAP



# O Limite da Simplicidade: Por que Variáveis Não Bastam

Até agora, manipulamos variáveis que armazenam um único valor, como um ID de cliente ou um preço. No mundo real, porém, os dados raramente andam sozinhos.

Uma entidade de negócio como um 'Cliente' é um conjunto de informações: Nome, Endereço, Telefone, Limite de Crédito.

lv_customer_id
1001

Variável Elementar: Uma Célula

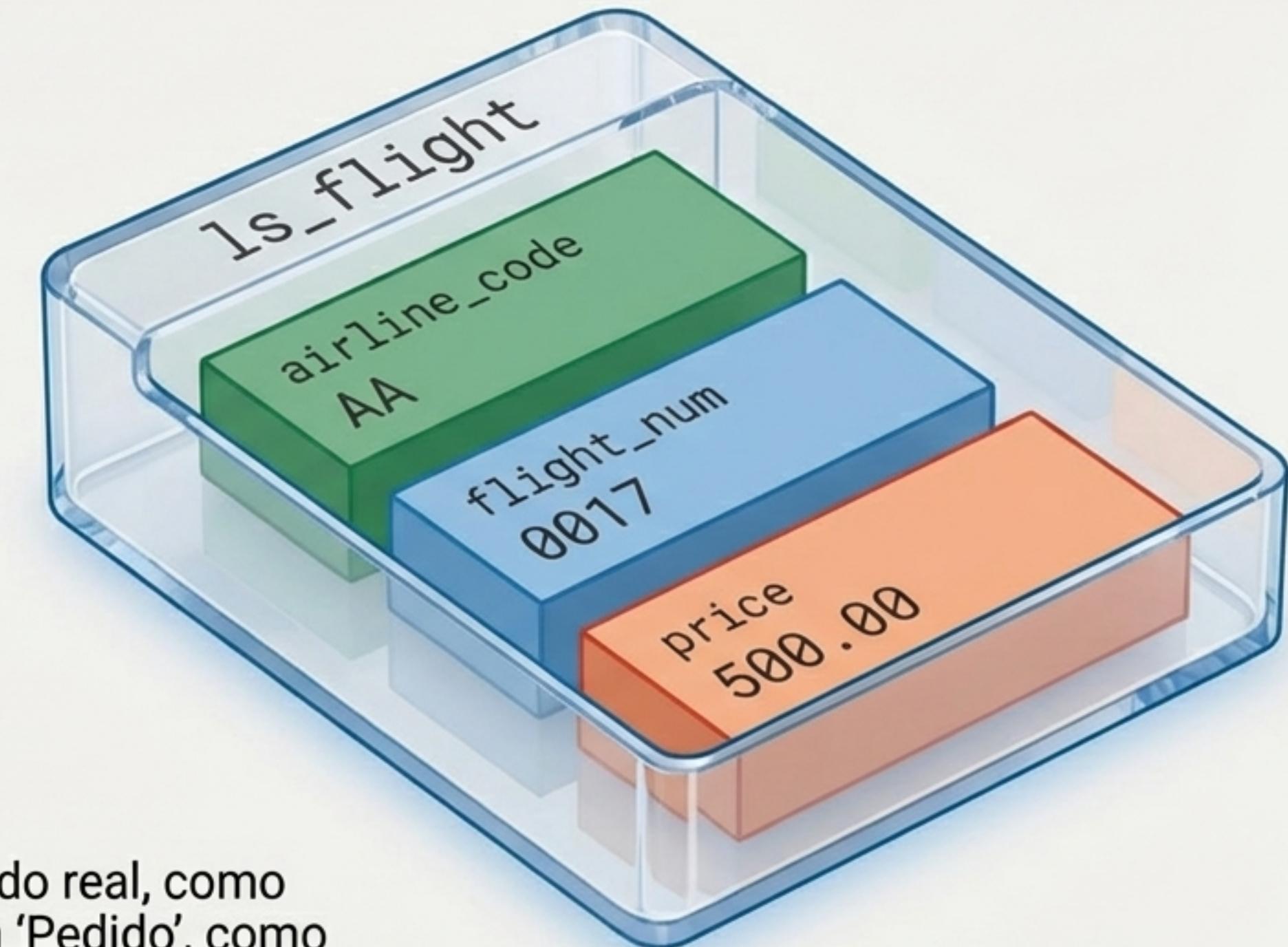
ID	Nome	Endereço	Limite
1001			

Entidade de Negócio: Uma Linha Inteira

# A Solução: Estruturas como Contêineres de Dados Lógicos

## Definição Central

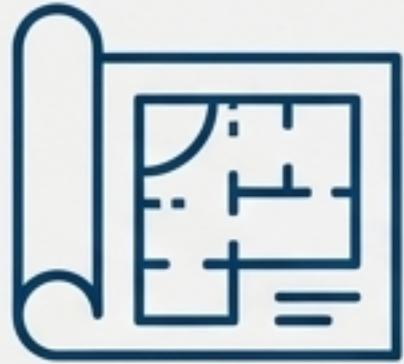
Uma **Estrutura** é a representação técnica de um agrupamento lógico. É uma área de memória contínua dividida em subáreas chamadas **componentes**.



## Contexto de Negócio

Ela nos permite tratar uma entidade do mundo real, como como um 'Voo' ou um 'Pedido', como um único 'Pedido', como uma única unidade de dados em nosso código.

# A Planta vs. A Construção: Entendendo TYPES e DATA



‘TYPES’ define o ‘molde’ da estrutura. Não aloca memória, apenas ensina ao compilador como os dados devem ser organizados.

```
TYPES: BEGIN OF ty_flight_info,  
        airline_code TYPE /dmo/carrier_id,  
        flight_num   TYPE /dmo/connection_id,  
        price       TYPE /dmo/flight_price,  
    END OF ty_flight_info.
```

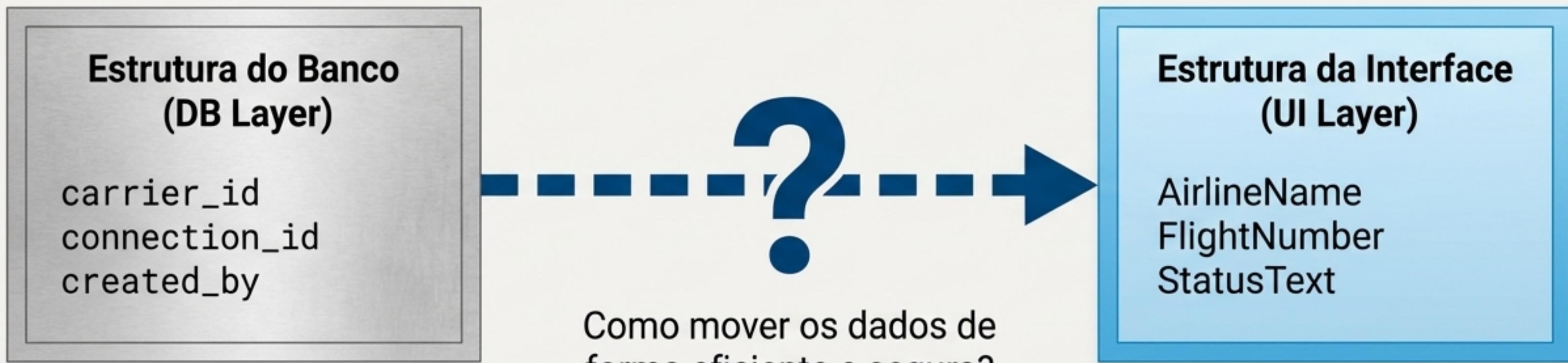


‘DATA’ aloca o espaço na memória e cria a ‘instância’ real e utilizável, pronta para armazenar valores.

```
" Cria a variável baseada no 'molde'  
DATA ls_flight TYPE ty_flight_info.  
  
" Acessa os componentes com hífen (-)  
ls_flight-airline_code = 'AA'.  
ls_flight-price      = '500.00'.
```

# O Desafio Moderno: A Jornada dos Dados Entre Camadas

No desenvolvimento RAP, dados fluem constantemente: do banco para a lógica de negócio, e da lógica para a interface do usuário. Cada camada pode ter sua própria "visão" da mesma entidade.

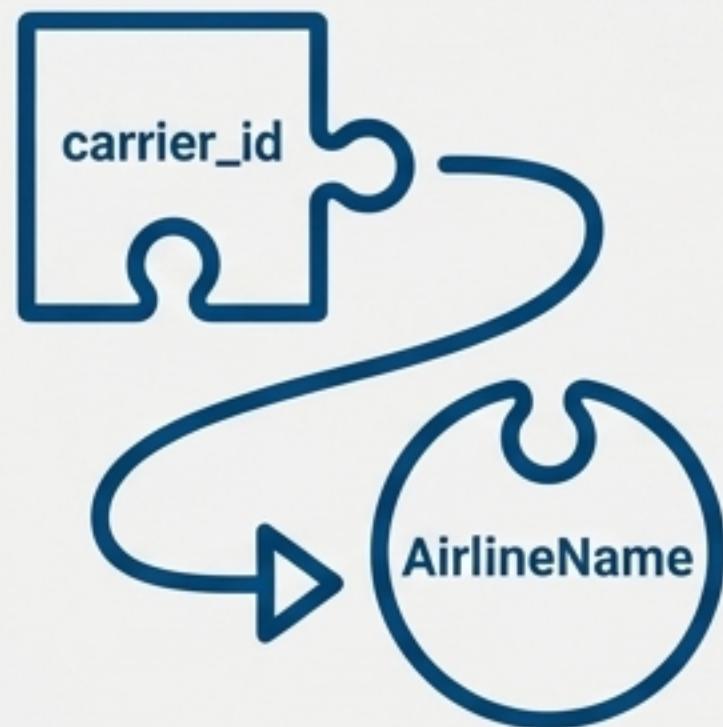


# A Ferramenta Essencial: O Operador `CORRESPONDING`

Copiar campo a campo é trabalhoso e propenso a erros. O operador `CORRESPONDING` resolve isso de forma inteligente, mapeando campos com nomes idênticos.

ABAP Legado (Evitar)	ABAP Moderno (Recomendado) 
<code>MOVE-CORRESPONDING ls_a TO ls_b.</code>	<code>ls_b = CORRESPONDING #( ls_a ).</code>
<b>**Riscos**:</b> Não limpa campos extras no destino, pode deixar "lixo" de dados.	<b>**Benefícios**:</b> Cria uma nova instância limpa, garante segurança de tipo e clareza de intenção.

# Dando Superpoderes ao 'CORRESPONDING': 'MAPPING' e 'EXCEPT'



**Problema:** E quando os nomes dos campos são diferentes, como `carrier\_id` e `AirlineName`?

**Solução:** Usamos `MAPPING` para criar um dicionário de tradução explícito dentro da operação.

```
MAPPING AirlineName  
= carrier_id
```



**Problema:** Como evitar que campos-chave ou técnicos (`created\_by`) sejam sobreescritos acidentalmente?

**Solução:** Usamos `EXCEPT` para listar os campos do destino que devem ser ignorados e protegidos durante a cópia.

```
EXCEPT created_by
```

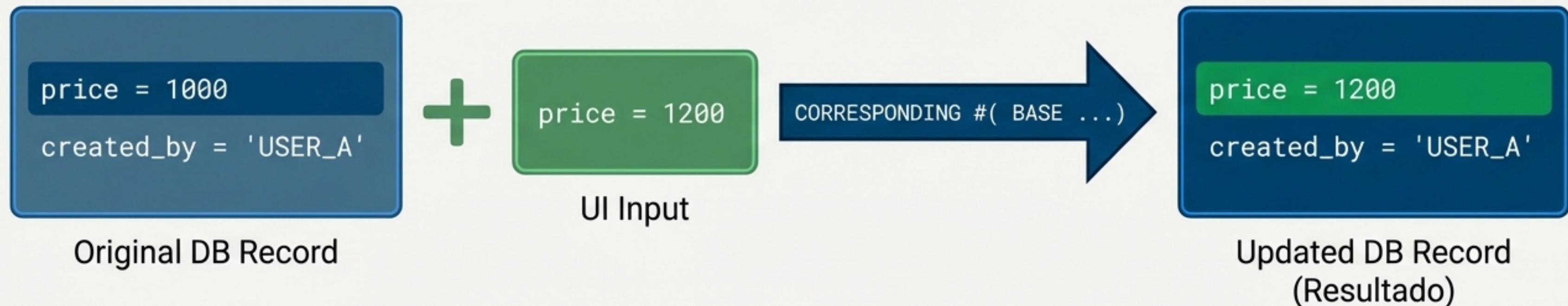
# A Arte da Atualização: Preservando Dados com `BASE`

## Cenário

Um usuário altera apenas o preço de um voo na tela. Como aplicamos essa mudança no banco de dados sem apagar os outros campos, como `created\_by` e `created\_at`?

## A Solução é `BASE`

`BASE` instrui o operador `CORRESPONDING` a usar uma estrutura existente como ponto de partida. Ele não cria uma nova estrutura do zero, mas sim aplica as mudanças por cima da base.



**\*\*Merge, Não Replace.\*\***

# Uma Nova Dimensão: Estruturas Planas vs. Profundas

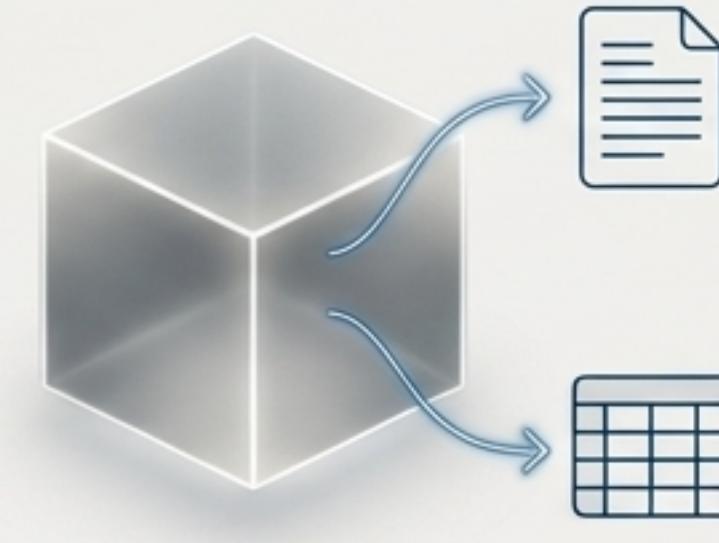
## Estrutura Plana (Flat Structure)



Contém apenas tipos elementares de tamanho fixo. Pense em uma linha de uma tabela de banco de dados simples.

Rápida e eficiente.

## Estrutura Profunda (Deep Structure)



Contém pelo menos um componente de tamanho dinâmico, como uma **String**, uma **Tabela Interna** ou uma **Referência de Objeto**.

Exige mais cuidado no manuseio da memória.

```
TYPES: BEGIN OF ty_passenger,  
        name    TYPE string,           " Torna a estrutura Deep  
        flights TYPE tt_flights,     " Tabela Interna  
    END OF ty_passenger.
```

# A Jornada Completa: Exemplo Prático (Parte 1 - As Definições)

Vamos simular um cenário RAP completo: ler dados do banco, prepará-los para a UI e, em seguida, processar uma atualização vinda da UI.

## Estrutura do Banco de Dados (`ty\_db\_flight`)

```
TYPES: BEGIN OF ty_db_flight,  
        carrier_id      TYPE string,  
        connection_id   TYPE string,  
        price           TYPE p LENGTH 10 DECIMALS 2,  
        created_by      TYPE string,  
        created_at      TYPE timestamp,  
    END OF ty_db_flight.
```

Nomes técnicos e campos de auditoria, espelhando a tabela do banco.

## Estrutura da Interface (`ty\_ui\_flight`)

```
TYPES: BEGIN OF ty_ui_flight,  
        airline         TYPE string,  
        connection     TYPE string,  
        price          TYPE p LENGTH 10 DECIMALS 2,  
        status_text    TYPE string,  
    END OF ty_ui_flight.
```

Nomes amigáveis para o usuário, sem campos de sistema. Inclui campos exclusivos da UI.

# A Jornada Completa: Exemplo Prático (Parte 2 – Lendo do DB para a UI)

**Objetivo da Operação:** Transformar a estrutura técnica do banco de dados em uma estrutura amigável para a interface do usuário.

1. Dados de origem (simulando um SELECT)

```
DATA(ls_db_source) = VALUE ty_db_flight( carrier_id = 'LH' price = '1250.50' ... ).
```

2. Transformação com MAPPING

```
DATA(ls_ui_target) = CORRESPONDING ty_ui_flight(  
    ls_db_source  
    MAPPING airline      = carrier_id  
          connection = connection_id  
).
```

Entrada (`ls\_db\_source`)

```
carrier_id = 'LH'  
price      = '1250.50'  
created_by = 'USER_SAP'
```



Saída (`ls\_ui\_target`)

```
airline      = 'LH'  
price      = '1250.50'  
status_text = ''
```

Observação: Note que `created\_by` é ignorado automaticamente, pois não existe no destino.

# A Jornada Completa: Exemplo Prático (Parte 3 – Atualizando o DB com Dados da UI)

**Objetivo da Operação:** Receber uma alteração de preço da UI e aplicá-la à estrutura do banco, preservando os campos de auditoria originais.

```
" 1. A UI envia um novo preço
DATA(ls_ui_input) = ls_ui_target.
ls_ui_input-price = '999.00'.

" 2. Merge inteligente usando BASE e MAPPING reverso
DATA(ls_db_updated) = CORRESPONDING ty_db_flight(
    BASE ( ls_db_source )                      " Ponto de partida: os dados atuais do DB
    ls_ui_input                                " Aplica as mudanças da UI
    MAPPING carrier_id = airline
    EXCEPT carrier_id                          " Protege o campo chave
).
```

## Análise do Resultado Final (`ls\_db\_updated`)

- ✓ `price`: '999.00' (✓ **Atualizado** com o valor da UI)
- ✓ `created\_by` : 'USER\_SAP' (✓ **Preservado** graças ao `BASE`)
- ✓ `carrier\_id` : 'LH' (✓ **Protegido** pelo `EXCEPT`)

# Guia Rápido: Modernize Seu Código ABAP

Adotar a sintaxe moderna não é apenas uma questão de estilo. É sobre **escrever código mais seguro, legível e robusto.**

Conceito	Abordagem Legada (Evitar)	Abordagem Moderna (Recomendado)
Transferência por Nome	MOVE-CORRESPONDING a T0 b.	b = CORRESPONDING #( a ).
Preservar Dados	Lógica manual de IFs para não limpar	b = CORRESPONDING #( BASE (b) a ).
Inicialização	Declaração linha a linha	DATA(s) = VALUE tipo( ... ).
Declaração de Work Area	DATA: wa LIKE ztabela.	DATA: wa TYPE ztabela.

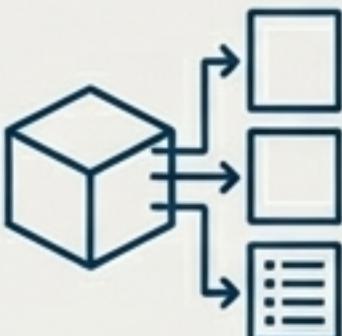
# Seu Glossário Técnico



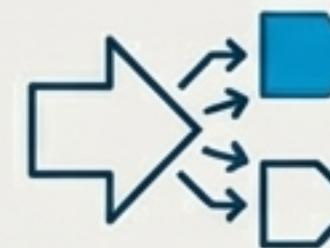
**Estrutura (Structure):** Objeto de dados complexo que agrupa componentes (campos), representando uma entidade de negócio.



**Estrutura Plana (Flat):** Contém apenas tipos elementares de comprimento fixo. Sem strings, tabelas internas ou referências.



**Estrutura Profunda (Deep):** Contém referências (strings, tabelas internas, objetos), exigindo gerenciamento de memória mais complexo.



**Operador CORRESPONDING:** Construtor que projeta dados de uma estrutura para outra com base em nomes ou regras de mapeamento.



**Adição BASE:** Cláusula que define um valor inicial para a estrutura de destino, essencial para operações de *Merge* ou *Update*.



**TYPES vs. DATA:** TYPES é o “molde” (compilação). DATA é a “instância” em memória (execução).

# Blocos de Construção para Aplicações Modernas

**Estruturas são os blocos de construção fundamentais para modelar o mundo dos negócios em código.**

O operador CORRESPONDING com suas variações (MAPPING, EXCEPT, BASE) não é apenas uma sintaxe, mas a ferramenta essencial para orquestrar a transformação de dados de forma segura e eficiente no modelo RAP.