

Decodificando CXN

A Arquitetura por Trás das Expressões CDS

Um guia visual para entender a notação interna do SAP Cloud Application Programming Model (CAP).

O Que é CXN e Por Que é Importante?

- CXN, ou *CDS Expression Notation*, é a representação interna, em formato JSON, de todas as expressões utilizadas em definições e queries CDS.
- É a linguagem que o compilador CDS usa para entender e processar sua lógica de negócio antes de traduzi-la para o banco de dados.



Compreender CXN permite depurar queries complexas, criar ferramentas customizadas e dominar o funcionamento interno do CAP.

Os Blocos de Construção do CXN

Toda expressão CXN é composta por um pequeno conjunto de blocos de construção em JSON. Vamos atribuir uma cor a cada um para facilitar a visualização.



val

Representa valores literais e estáticos. (ex: 123, 'texto', true)



ref

Representa referências dinâmicas a colunas ou entidades.



xpr

O tecido conectivo que une operadores e operandos.



func

Representa chamadas de função e transformações.



list

Agrupa múltiplos elementos em uma coleção ou lista.

O Fundamento: Valores Literais com `val: ...`

Expressão CDS

```
'a string'  
11  
true  
null  
date'2023-04-15'  
timestamp'2023-04-15T13:05:23Z'
```

Representação CXN

```
{val: 'a string'}  
{val: 11}  
{val: true}  
{val: null}  
{val: '2023-04-15', literal: 'date'}  
{val: '2023-04-15T13:05:23Z',  
literal: 'timestamp'}
```

A chave `val` contém o valor literal exatamente como especificado em JSON. Literais tipados como `date` ou `timestamp` incluem uma propriedade adicional.

Apontando para Dados: Referências com `'{ref: [...]}'`

Uma referência (`ref`) aponta para um elemento de dados, como uma coluna ou uma associação. É representada por um array de segmentos de caminho.

Expressão CDS

`![keyword]`

`foo.bar`

Representação CXN

`{"ref": ['keyword']}`

`{"ref": ['foo', 'bar']}`

Referências Avançadas: Adicionando Filtros e Argumentos

A estrutura de `'{ref: [...]}'` pode ser expandida. Um segmento pode ser um objeto para incluir cláusulas como `where`, `groupBy`, `orderBy` e `limit`.

```
foo[where a=9 group by b having b>2 order by c limit 7].bar
```

```
{  
  ref: [  
    {  
      id: 'foo',  
      where: [[{ref: ['a']}, '=', {val: 9}],  
      groupBy: [[{ref: ['b']}],  
      having: [[{ref: ['b']}, '>', {val: 2}],  
      orderBy: [[{ref: ['c']}]],  
      limit: {rows: {val: 7}}  
    },  
    'bar'  
  ]  
}
```

Executando Ações: Chamadas de Função com `'{func: ...}'`

Funções são representadas com a chave `func` para o nome da função e `args` para seus argumentos, que podem ser um array (posicional) ou um objeto (nomeado).

Expressão CDS

sum(x)

foo(p => x)

count(*)

rank() over (...)

Representação CXN

{func: 'sum', args: [{ref: ['x']}]}

{func: 'foo', args: {p: {ref: ['x']}}}

{func: 'count', args: ['*']}

{func: 'rank', args: [], xpr: ['over', {xpr:[...]}]}

Note que a cláusula `over` de funções de janela SQL é capturada usando um `xpr`, demonstrando como os blocos se compõem.

Casos Especiais: Funções como Operadores em {xpr: ...}

Atenção: chamadas de função no estilo 'método' (`.`) e instanciações com `new` não são representadas por {func: ...}. Em vez disso, são tratadas como expressões de operador (`xpr`), onde `.` e `new` são os operadores.

Expressão CDS

```
shape.ST_Area()
```

Representação CXN

```
{xpr: [[{ref: ['shape']}, '.', {func: 'ST_Area', args: []}]]}
```

```
new ST_Point(2, 3)
```

```
{xpr: ['new', {func: 'ST_Point', args: [{val: 2}, {val: 3}]}]}
```

Isso mostra a flexibilidade do `xpr` como o principal mecanismo para combinar elementos em CXN.

A Cola do Sistema: Expressões de Operador com {xpr: ...}

O bloco {xpr: ...} é o mais versátil. Ele une um ou mais operandos com operadores para formar expressões complexas.

Sua estrutura é um array simples contendo uma sequência de operandos (qualquer bloco CXN) e operadores (strings como `'=` ou 'and').

Expressão CDS

```
x < 9
```

Representação CXN

```
{xpr: [{ref: ['x']}, '<', {val: 9}]}
```

Parênteses `(...)` em uma expressão são representados por um {xpr: ...} aninhado, preservando a precedência.

Desconstruindo um {xpr: ...} Aninhado

Expressão CDS

```
x < 9 and (y = 1 or z = 2)
```

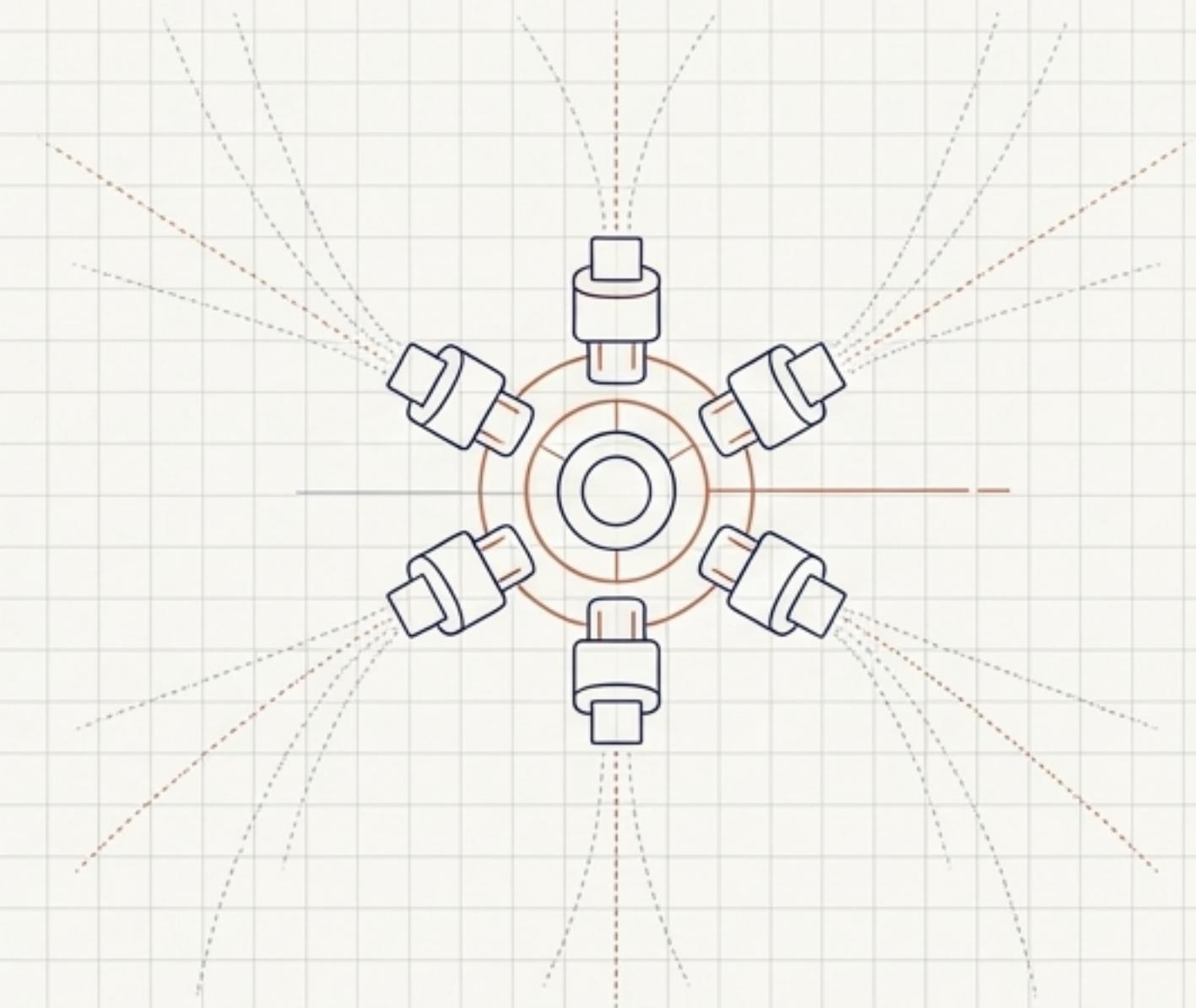
Representação CXN

```
{  
  xpr: [  
    {{ref: ['x']}, '<',  
     {{val: 9},  
      'and',  
      { xpr: [  
        {{ref: ['y']}, '=',  
         {{val: 1}, 'or',  
          {{ref: ['z']}, '='}, {{val: 2}}}  
        ] }  
      ] }  
  ]}
```

A Filosofia do **xpr**: "Ignorância Inteligente"

O CQN intencionalmente não tenta entender os operadores individuais (`and`, `or`, `=`). Ele apenas os captura como uma sequência, na mesma ordem em que aparecem no código fonte.

- "Essa 'ignorância' nos permite permanecer abertos a qualquer tipo de operador e palavra-chave. Por exemplo, podemos expressar facilmente extensões nativas de dialetos de banco de dados subjacentes."



Key Takeaway: Essa abordagem garante que o CAP seja extensível e à prova de futuro, capaz de suportar novos recursos de banco de dados sem exigir alterações no compilador principal.

Uma Exceção à Regra: O Operador Ternário

Como uma exceção à regra de 'ignorância', o CDS suporta o operador ternário (`?:`) no código fonte, mas o converte imediatamente para uma expressão `CASE` equivalente em CXN.

Código CDS

```
x < 10 ? y : z
```

Blueprint CXN

```
{
  "xpr": [
    "case", "when", {"ref": ["x"]}, "<", {"val": 10},
    "then", {"ref": ["y"]},
    "else", {"ref": ["z"]},
    "end"
  ]
}
```

Isso simplifica o processamento interno, mapeando uma construção de alto nível para uma expressão SQL mais universal.

Agrupando Elementos: Listas e Tuplas com {list: ...}

Para representar uma coleção de valores, como em uma cláusula `IN` ou uma tupla, use o bloco `{list: ...}`. Ele contém um array de expressões.

Código CDS

```
(1, 2, 3)  
(foo, bar)
```

Blueprint CXN

```
{list: [{val: 1}, {val: 2}, {val: 3}]}  
{list: [{ref: ['foo']}, {ref: ['bar']}]}
```

Queries Dinâmicas: Parâmetros de Ligação (`param`)

Parâmetros para *prepared statements* são representados por um bloco `'{ref: [...]}'` com uma propriedade adicional `param: true`.

Código CDS

x = ?

x = :1

x = :y

Blueprint CXN

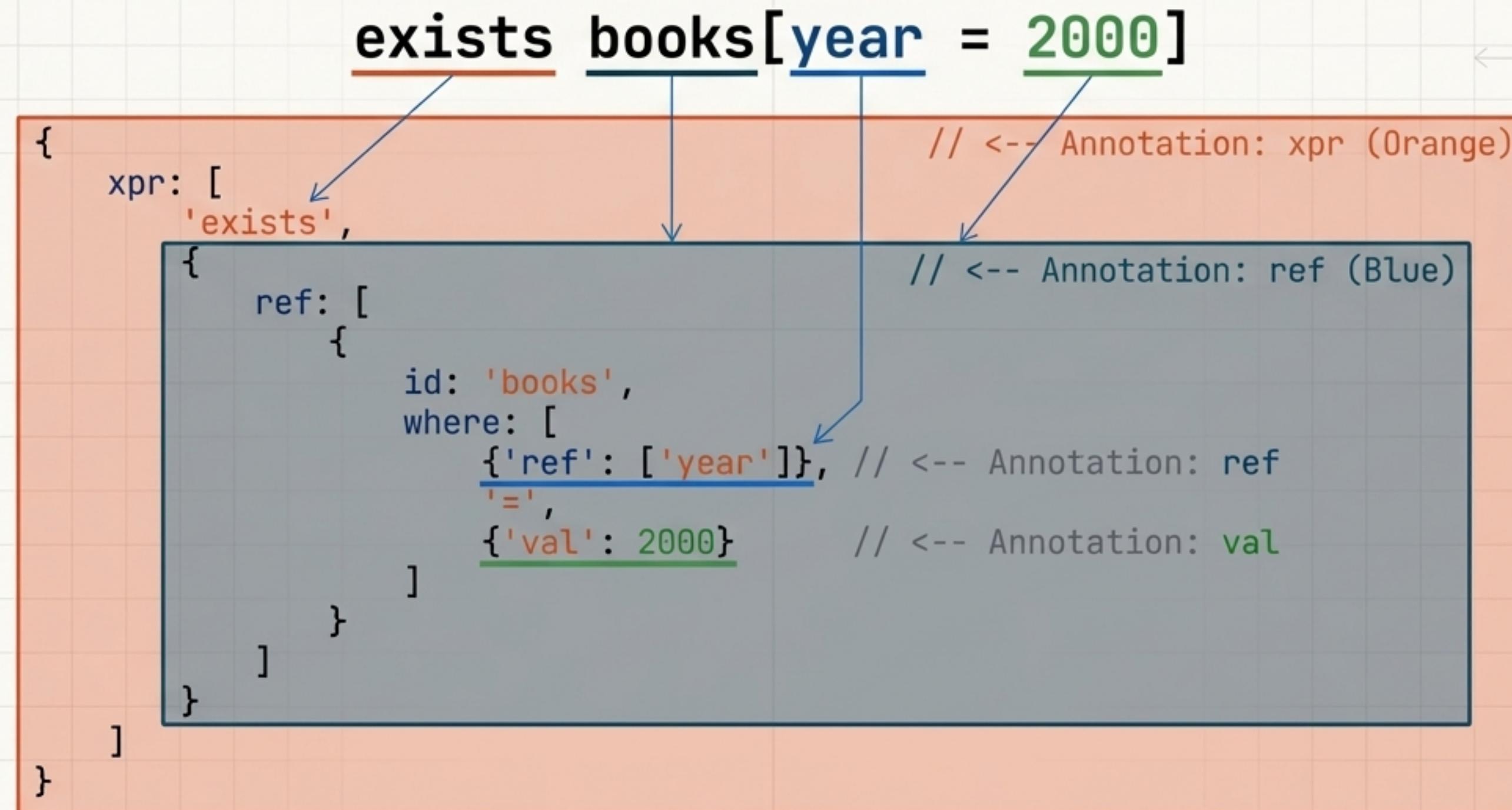
```
{xpr: [{ref: ['x'], '='}, {ref: ['?'], param: true}]}
{xpr: [{ref: ['x'], '='}, {ref: ['?'], param: true}]}

{xpr: [{ref: ['x'], '='}, {ref: [1], param: true}]}

{xpr: [{ref: ['x'], '='}, {ref: ['y'], param: true}]}
```

A Arquitetura Completa: Juntando Todos os Blocos

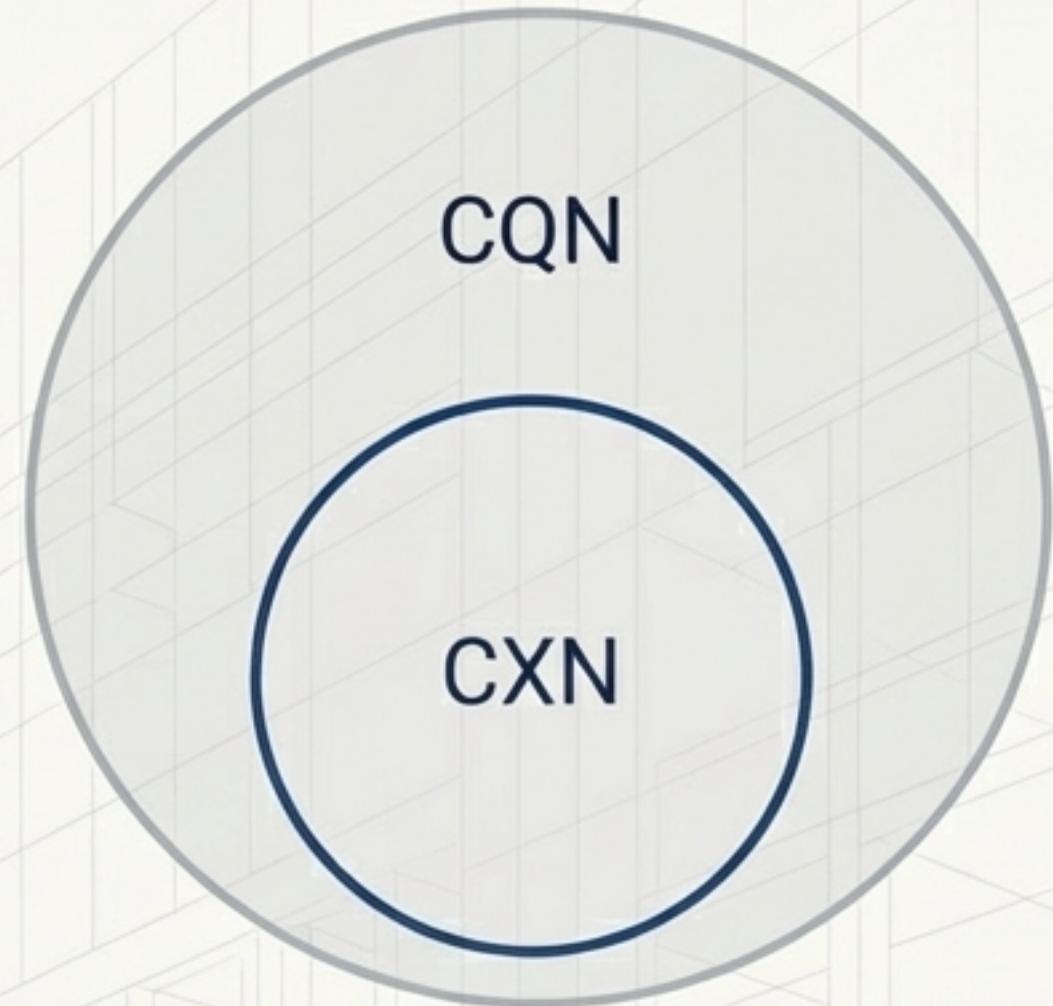
Vamos analisar uma expressão complexa e ver como nossos blocos de construção (`xpr`, `ref`, `val`) se combinam.



Próximos Passos: Expressões e Além

Cobrimos os blocos de construção fundamentais do CXN, a linguagem das expressões em CDS.

- **Subqueries:** Expressões `SELECT` completas também podem ser aninhadas, formando um bloco próprio em CXN.
- **CQN (CDS Query Notation):** O CXN é parte do CQN, a notação completa para queries `SELECT`, `INSERT`, `UPDATE` e `DELETE`.



Para explorar a estrutura completa de queries, incluindo projeções, fontes de dados e operações de modificação, consulte a documentação oficial do **Core / Query Notation (CQN)**.