

Dominando o `cds.DatabaseService`

Um Guia Prático de Configuração e Boas
Práticas para Desenvolvedores CAP Node.js

Nossa Jornada



Fundamentos: O que é o 'DatabaseService' e como ele gerencia transações.



Interações Essenciais: Entendendo os resultados de operações, como o 'InsertResult'.



Performance e Estabilidade: O guia definitivo para configurar o Connection Pooling.



Capacidades Avançadas: Usando a ferramenta UPSERT' для replicação de dados.

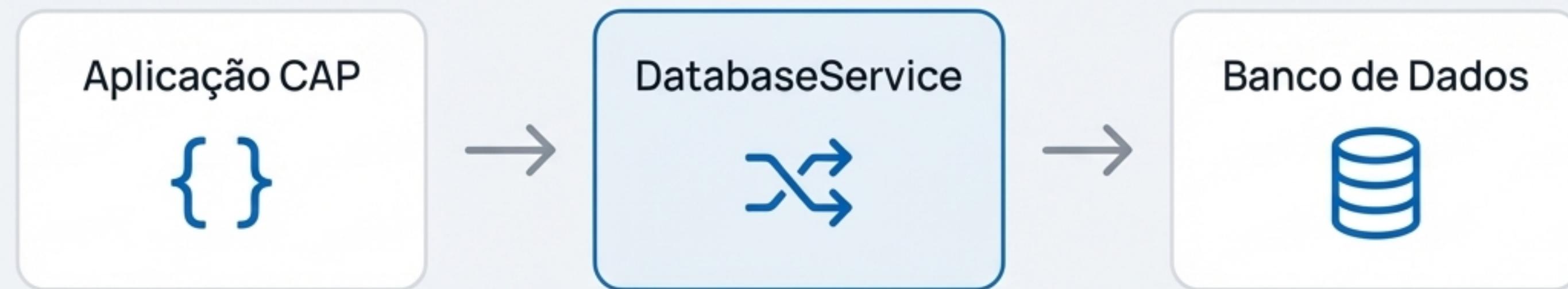


Principais Takeaways: O que você precisa lembrar.

O Que É o `cds.DatabaseService`?

Classes `cds.DatabaseService` e suas subclasses são serviços técnicos que representam o **armazenamento persistente**.

Elas atuam como a camada de abstração entre a sua lógica de aplicação CAP e o banco de dados físico.



Fundamentos

O Ciclo de Vida da Transação: O Papel do `.begin()`

O método `$srv.begin()` obtém uma conexão do pool e pode iniciar uma transação no banco de dados (ex: BEGIN TRANSACTION).



O Framework Gerencia Para Você

Este método é chamado automaticamente pelo framework na primeira query.
Você nunca precisa chamá-lo diretamente no código da sua aplicação.

Casos de uso para chamadas manuais são extremamente raros e **não são considerados uma boa prática** (por exemplo, reutilizar um objeto tx após um commit ou rollback).

Entendendo os Resultados de `INSERT` com `InsertResult` (Beta)

Em operações de `INSERT`, os `DatabaseServices` retornam uma instância de `InsertResult`.

Iterador:	Retorna as chaves das entradas criadas. [...result] -> [{ ID: 1 }, { ID: 2 }, ...]
	Nota: No caso de `INSERT...as(SELECT...)`, o iterador retorna `{}` para cada linha.
`affectedRows`:	O número de entradas (raiz) inseridas ou o número de `affectedRows` no caso de `INSERT into SELECT`.
`valueOf()`:	Retorna `affectedRows`, permitindo comparações diretas como `result > 0`.



Use `result > 0` para checagens. O operador `==` não funcionará como esperado, pois ele também compara o tipo do objeto.

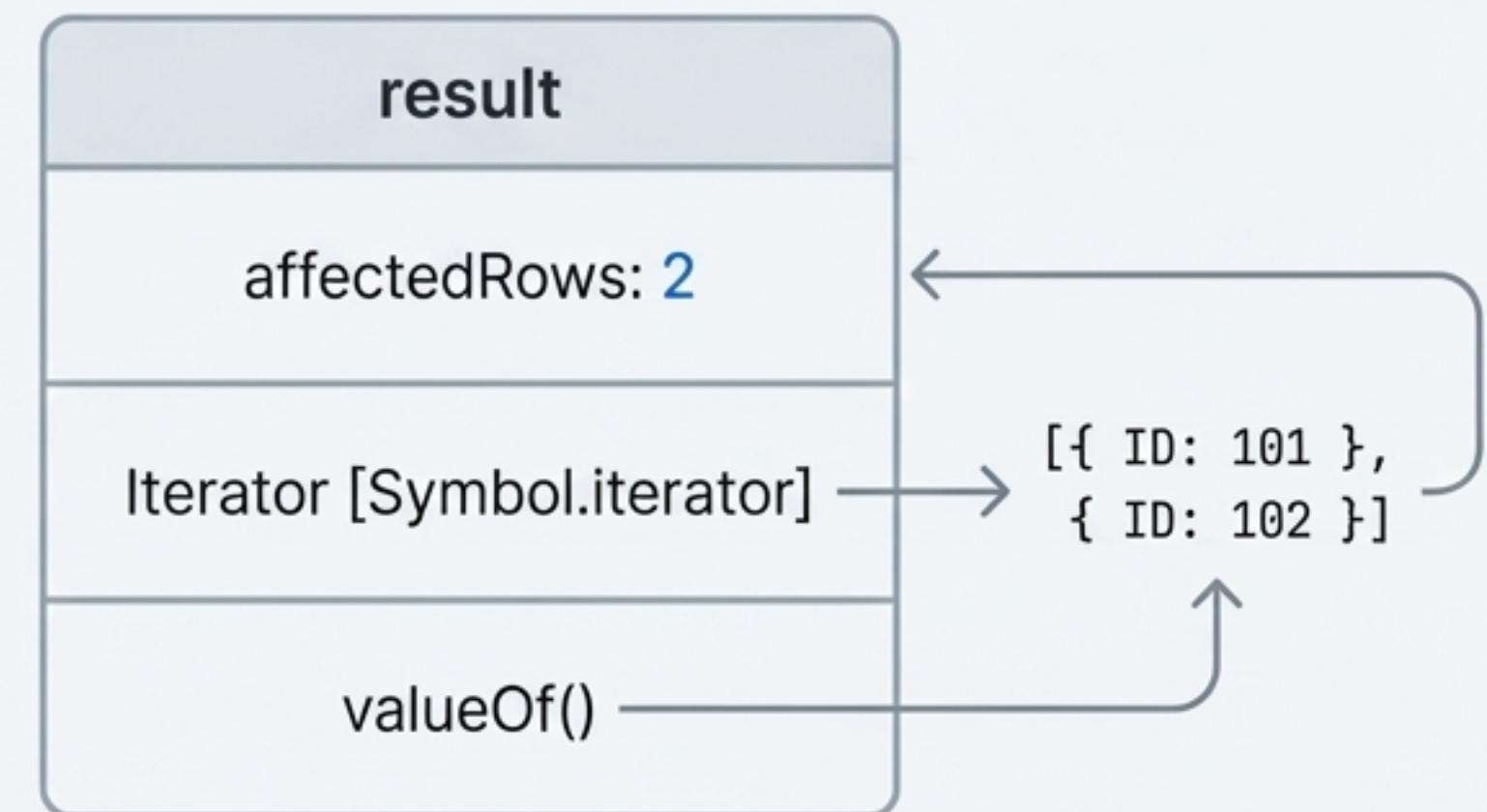
Interações Essenciais

‘InsertResult’ na Prática: Do Código ao Objeto

‘INSERT’ Query

```
const data = [{ name: 'Book A' }, { name: 'Book B' }];
const result = await INSERT.into(Books).entries(data);
```

Objeto ‘InsertResult’



Performance e Estabilidade

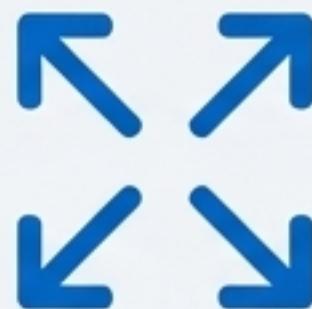
A Base da Performance: Connection Pooling

Em vez de abrir e fechar uma conexão com o banco de dados para cada requisição, o CAP utiliza um pool baseado no generic-pool para reutilizar conexões.



Eficiência

Reduz a sobrecarga de estabelecer novas conexões.



Escalabilidade

Gerencia um número máximo de conexões simultâneas, evitando sobreregar o banco de dados.



Resiliência

O 'evictor' integrado inspeciona e remove conexões ociosas ou antigas, mantendo o pool saudável.

Painel de Controle: Configurando Seu Pool de Conexões

Ajustar o pool é essencial para a performance em produção. Estes são os parâmetros chave que você pode configurar.



Dimensionamento do Pool (Sizing)

- min
- max



Tempo de Vida da Conexão (Lifetime)

- softIdleTimeoutMillis
- idleTimeoutMillis



Comportamento de Aquisição (Acquisition)

- acquireTimeoutMillis
- fifo
- testOnBorrow



Manutenção do Pool (Eviction)

- evictionRunIntervalMillis
- numTestsPerEvictionRun

Performance e Estabilidade

Ajuste Fino: Dimensionamento e Aquisição

`max`

Número máximo de conexões a serem mantidas no pool.

`min`

Número mínimo de conexões a serem mantidas no pool.

`acquireTimeoutMillis`

Tempo máximo (ms) de espera para obter uma conexão do pool.

`fifo`

Se `true`, aloca as conexões mais antigas (fila). Se `false` (padrão), aloca as mais recentes (pilha).



`min`

Mantenha o valor padrão de '0'.

Configurar `min > 0` faz com que, a cada ciclo de 'eviction', o pool destrua conexões ociosas e imediatamente crie novas conexões até atingir o mínimo, gerando sobrecarga desnecessária.

Ajuste Fino: Manutenção e Tempo de Vida das Conexões

evictionRunIntervalMillis	Frequência (ms) com que a verificação de 'eviction' é executada. `0` desativa a verificação.
numTestsPerEvictionRun	Quantidade de conexões a serem verificadas em cada ciclo de 'eviction'.
softIdleTimeoutMillis	Tempo (ms) que uma conexão pode ficar ociosa antes de ser elegível para 'eviction'. O pool tentará manter pelo menos `min` conexões.
idleTimeoutMillis	Tempo mínimo (ms) que uma conexão pode ficar ociosa antes de ser elegível para 'eviction'. Este parâmetro substitui `softIdleTimeoutMillis` .
testOnBorrow	Valida as conexões antes de entregá-las aos clientes?

Performance e Estabilidade

Ponto Crítico: A Configuração do Pool é Sua Responsabilidade



Os parâmetros são altamente específicos para cada ambiente técnico (ambiente da aplicação, localização do banco de dados, etc.).

Apesar de fornecermos uma configuração padrão, esperamos que cada aplicação forneça sua própria configuração com base em suas necessidades específicas.

```
{  
  "cds": {  
    "requires": {  
      "db": {  
        "kind": "hana",  
        "pool": {  
          "acquireTimeoutMillis": 5000,  
          "min": 0,  
          "max": 100,  
          "fifo": true  
        }  
      }  
    }  
  }  
}
```

A Ferramenta de Replicação: Conhecendo o `UPSERT`

- O principal caso de uso do `UPSERT` é a **replicação de dados**.
- Ele **atualiza** registros existentes a partir dos dados fornecidos ou **insere** novos registros se eles não existirem no banco de dados.



Mesmo que uma entidade não exista no banco de dados:
`UPSERT` não é equivalente a `INSERT`.

```
await UPSERT.into('db.Books').entries(data)
```

As Regras do 'UPSERT'

- ✓ **Chaves são Mandatórias:** Os dados do `UPSERT` devem conter todos os elementos da chave da entidade.
- ✓ **Sem `WHERE` Clause:** Os valores da chave para a operação são extraídos diretamente dos dados fornecidos.
- ✓ **Semântica de 'PATCH':** Se os dados estiverem incompletos, apenas os valores fornecidos são atualizados ou inseridos.
- ✓ **Estrutura Consistente:** Todas as linhas nos dados a serem inseridos/atualizados precisam ter a mesma **estrutura** (mesmos campos nomeados).

Dica Técnica

Queries `UPSERT` são traduzidas para comandos nativos do banco:

- **SAP HANA:** `UPSERT`
- **SQLite:** `INSERT ON CONFLICT`

Capacidades Avançadas

Limitações e Pontos de Atenção do `UPSERT`

As seguintes ações **não são executadas** durante uma operação de `UPSERT`:

- ✗ **Geração de UUIDs:** Valores de chave do tipo UUID **não são gerados** automaticamente.
- ✗ **Invocação de Handlers CAP:** Handlers genéricos do CAP, como o de 'audit logging', **não são invocados**.



Diferença para o Runtime Java

Em contraste com o runtime Java, deep upserts e delta payloads ainda não são suportados na versão Node.js.

Principais Takeaways para Seu Próximo Projeto

1

Deixe as Transações com o Framework

Não chame `.begin()` manualmente. O CAP gerencia o ciclo de vida da transação para você.

2

Sua Configuração de Pool é Essencial

A configuração padrão não serve para todos. Analise e ajuste os parâmetros do pool (`min`, `max`, `timeouts`) para seu cenário de produção. Lembre-se: `min: 0`.

3

UPsert é uma Ferramenta Específica

Use-o para replicação de dados, mas esteja ciente de suas limitações (sem geração de UUID, sem handlers CAP, sem deep upserts no Node.js).