Documentação Técnica: Transformação Silver (dbt)

Status: ✅ Fase Silver Concluída **Data:** 30/01/2026

1. Objetivo

Configurar o ambiente de transformação de dados (dbt) e criar a camada **Silver**, responsável por limpar, padronizar e remover duplicatas dos dados brutos ingestados na camada Bronze.

2. Instalação e Configuração

2.1. Instalação do Adaptador

Para que o dbt converse com o Supabase (Postgres), instalamos o adaptador específico no ambiente virtual:

PowerShell:

pip install dbt-postgres

2.2. Inicialização do Projeto

Criamos a estrutura padrão do dbt:

PowerShell

dbt init transformacao_btc

obs:(dar um CLT + C para interromper)

PowerShell

cd transformacao_btc

2.3. Configuração de Conexão (profiles.yml)

Criamos manualmente o arquivo profiles.yml na raiz da pasta transformacao_btc. Atenção: Utilizamos as configurações do Supabase Pooler (porta 6543) para garantir estabilidade na conexão.

YAML

```yaml
transformacao_btc:
  target: dev
  outputs:
    dev:
      type: postgres
      host: aws-0-sa-east-1.pooler.supabase.com  # Host do Pooler (sem https)
      user: postgres.seu_projeto_id              # Formato obrigatório: postgres.
      password: [SUA_SENHA_DO_BANCO]
      port: 6543                                 # Porta do Pooler (não é 5432)
      dbname: postgres
      schema: public
      threads: 1
```

2.4. Ajuste do Projeto (dbt_project.yml)

Limpamos o arquivo dbt_project.yml para remover referências a exemplos e configurar o schema da camada Silver.

YAML

```yaml
name: 'transformacao_btc'
version: '1.0.0'
config-version: 2

profile: 'transformacao_btc'

model-paths: ["models"]
analysis-paths: ["analyses"]
test-paths: ["tests"]
seed-paths: ["seeds"]
macro-paths: ["macros"]
snapshot-paths: ["snapshots"]

target-path: "target"
clean-targets:
  - "target"
  - "dbt_packages"

models:
  transformacao_btc:
    silver:
      +materialized: view
      +schema: silver
    bronze:
      +materialized: view
      +schema: bronze
```

3. Modelagem de Dados (Código SQL)

3.1. Mapeamento da Fonte (models/bronze/sources.yml)

Informamos ao dbt onde encontrar a tabela bruta criada pelo Python.

YAML

```yaml
version: 2

sources:
  - name: supabase_bronze
    database: postgres
    schema: public
    tables:
      - name: bronze_bitcoin
```

3.2. Modelo Silver (models/silver/stg_bitcoin.sql)

Criamos a lógica de saneamento.

- **Tipo:** View (Virtual).
- **Tratamento:** Deduplicação via ROW_NUMBER(). Mantém apenas a ingestão mais recente de cada data.

```
{{ config(
  materialized='view',
  schema='silver'
) }}

WITH source_data AS (
  SELECT
    coin_id,
    price_usd,
    updated_at,
    ingestion_at,
    -- Cria ranking: 1 = Dado mais novo, 2+ = Duplicatas
    ROW_NUMBER() OVER(PARTITION BY updated_at ORDER BY ingestion_at DESC) as row_num
  FROM {{ source('supabase_bronze', 'bronze_bitcoin') }}
)

SELECT
```

coin_id,

price_usd,

updated_at,

ingestion_at

FROM source_data

WHERE row_num = 1

4. Execução e Testes

4.1. Teste de Conexão

Para verificar se o profiles.yml está correto:

PowerShell

```
dbt debug --profiles-dir .
```

(Resultado esperado: `Connection test: [OK connection ok]`)

4.2. Rodar a Transformação

Para criar a view Silver no banco de dados:

PowerShell

```
dbt run --profiles-dir .
```

(Resultado esperado: `OK created sql view model public_silver.stg_bitcoin`)

Obs: caso retorne erro:

Limpeza Profunda (dbt clean)

Vamos forçar o dbt a esquecer tudo o que ele lembra e recompilar do zero. No terminal, rode:

PowerShell

```
dbt clean
```