

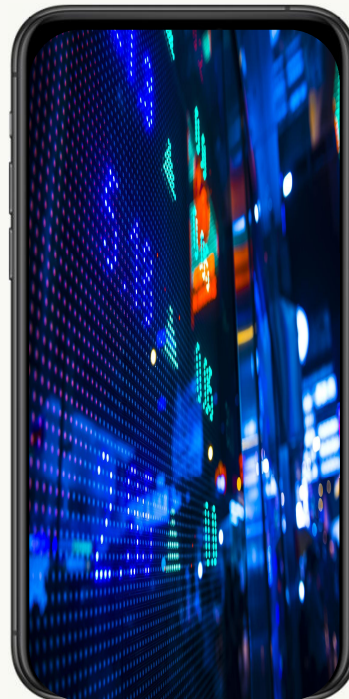
ESTRATÉGIAS DE PARTICIONAMENTO



Volume de Dados



Baixa Cardinalidade



Tamanho da Partição



Seleção de Colunas

Particionamento para Performance e Redução de Custos

Dataset de exemplo

Tabela: vendas (Delta Lake)

data	categoria	id_produto	valor
01/05/2025	Eletrônicos	1001	120
01/05/2025	Eletrônicos	1002	150
01/05/2025	Casa	2001	80
02/05/2025	Casa	2002	200
02/05/2025	Esportes	3001	90
03/05/2025	Eletrônicos	1003	110
03/05/2025	Casa	2003	70
03/05/2025	Esportes	3002	130
...	...	...	...

Dados físicos SEM particionamento

Todos os dados ficam misturados em poucos arquivos grandes.

Pasta da tabela (sem particionamento)

parte-0001.parquet 1.2 GB parte-0002.parquet 1.1 GB parte-0003.parquet 1.3 GB parte-0004.parquet 1.0 GB ...

Quando o particionamento é útil?

- ✓ Tabelas grandes com filtros frequentes em colunas de particionamento (ex.: data, país, categoria).
- ✓ Consulta sempre um subconjunto dos dados.
- ✓ Ingestão incremental (por dia, mês, etc.).
- ✓ Permite Data Skipping eficiente (ler apenas as partições necessárias).

Problema

- Para filtrar por data (ex.: data = '03/05/2025'), é preciso ler todos os arquivos.
- Mais I/O desnecessário e maior consumo de recursos.
- Consultas mais lentas e custo mais alto.

1 Como funciona o particionamento

- 1 Escolha as colunas para particionar
- 2 Grava os dados separados por valor da coluna
- 3 Cada partição contém apenas os dados daquele valor
- 4 Leituras mais eficientes

Exemplo: particionar por data (dia)

Motivo: ✓ Consultas sempre filtram por data (ex.: onde data = '03/05/2025')

Arquivos de exemplo:

- vendas/
- data=2025-05-01/
- data=2025-05-02/
- data=2025-05-03/
- ...

Partições de exemplo:

- data=2025-05-01 ~300 MB
- data=2025-05-02 ~350 MB
- data=2025-05-03 ~320 MB

Leituras mais eficientes

- ✓ O Spark lê apenas as partições necessárias para o filtro.
- ✓ Menos dados lidos = mais rápido e mais barato.
- ✓ Data Skipping funciona muito melhor.

2 Antes e depois do particionamento

Antes (sem particionamento)

Filtro da consulta: WHERE data = '03/05/2025'

Arquivos lidos pelo Spark

parte-0001.parquet 1.2 GB parte-0002.parquet 1.1 GB parte-0003.parquet 1.3 GB parte-0004.parquet 1.0 GB ...

Total lido: ~4.6 GB (todos os arquivos)

Depois (com particionamento por data)

Mesma consulta: WHERE data = '03/05/2025'

Partições avaliadas pelo Spark

data=2025-05-01 (pulado) data=2025-05-02 (pulado) data=2025-05-03 ~320 MB

Total lido: ~320 MB (apenas 1 partição)
Redução de ~93% de dados lidos!

Por que melhorou?

- ✓ O Spark identificou que apenas a partição data=2025-05-03 pode conter dados do filtro.
- ✓ As demais partições foram puladas sem leitura.
- ✓ Menor I/O, menos CPU, menos custo e resposta muito mais rápida!

3 Resumo rápido

Aspecto	Sem particionamento	Com particionamento (por data)
Organização dos dados	Tudo misturado em poucos arquivos	Dados separados por valor da coluna
Leitura de dados	Lê todos os arquivos	Lê apenas as partições necessárias
Data Skipping	Pouco eficiente	Muito eficiente
Performance das consultas	Mais lenta	Mais rápida
Custo (I/O e processamento)	Mais alto	Mais baixo
Manutenção / gestão	Simple	Requer escolha correta das colunas

★ Dica de ouro

Particione por colunas com:

- Alta cardinalidade temporal (data, mês, ano)
- Frequentemente usadas em filtros
- Baixo número de valores por partição

Evite:

- Colunas com muitos valores únicos (ex.: id_cliente)
- Muitas colunas de particionamento
- Partições muito pequenas (arquivos pequenos)

4 Boas práticas e cuidados

1. Cole o particionamento certo
 - ✓ Use colunas de filtro frequente (ex.: data).
 - ✓ Evite over-partitioning (muitas partições pequenas).
2. Monitore o tamanho das partições
 - ✓ Busque arquivos entre 128MB e 1GB (ideal para o Spark).
3. Combine com outras otimizações
 - ✓ Use particionamento + ZORDER para ainda mais eficiência.
4. Esquema recomendado
 - Exemplo de gravação particionada
 - ```
df.write\n .format("delta")\n .mode("append")\n .partitionBy("data")\n .save("/mnt/delta/vendas")
```

**Importante:** O particionamento é uma estratégia física de organização dos dados. Escolhido corretamente, é uma das formas mais eficazes de acelerar consultas e reduzir custos no lakehouse.