Projeto databricks lakehousecase

Lakehouse Case — Engenharia de Dados

Case técnico desenvolvido em Databricks Community Edition.
O projeto cobre ingestão, padronização, tratamento de qualidade, modelagem dimensional e exposição analítica a partir de 9 fontes heterogêneas.

Objetivo

Construir um pipeline Lakehouse completo utilizando PySpark + Delta Lake seguindo a arquitetura:

```
RAW → Bronze → Silver → Gold → BI
```

O foco principal do projeto foi:

- ingestão confiável
- padronização de dados inconsistentes
- rastreabilidade
- reprocessamento idempotente
- modelagem analítica
- qualidade de dados
- exposição semântica para BI

Arquitetura

```
SOURCES (9 arquivos)
    │
    ├──────────────────────────────────► CONTROL  (ingestion_log)
    ▼
BRONZE   — ingestão raw, schema explícito, colunas técnicas
    │
    ▼
SILVER   — limpeza, normalização, deduplicação, isolamento de inválidos/órfãos
    │
    ▼
GOLD     — modelo dimensional + métricas pré-agregadas + semantic layer
    │
    ▼
BI       — Databricks SQL conectado ao Gold
```

A camada `CONTROL` é escrita pelos notebooks Bronze e registra eventos de ingestão.
Não participa diretamente do fluxo analítico.

Cada camada pode ser reprocessada independentemente via `overwrite`.

Estrutura do repositório

```
notebooks/
├── 00_setup.py
├── bronze/
│   ├── 01_bronze_pedidos_cabecalho.py
│   ├── 02_bronze_pedidos_itens.py
│   ├── 03_bronze_cadastros.py
│   └── 04_bronze_json_sources.py
│
├── silver/
│   ├── 01_silver_pedidos_cabecalho.py
│   ├── 02_silver_pedidos_itens.py
│   ├── 03_silver_cadastros.py
│   └── 04_silver_json_sources.py
│
└── gold/
    ├── 01_gold_dimensoes.py
    ├── 02_gold_fato_pedidos.py
    ├── 03_gold_metricas.py
    └── 04_gold_vw_kpis_pedidos.py

tests/
├── validacoes_pos_carga.py
└── data_quality_checks.sql

docs/
├── qualidade_dados.md
└── resumo_executivo.md

sql/
└── consultas_bi.sql
```

Tecnologias utilizadas

- Databricks Community Edition
- PySpark
- Delta Lake
- Databricks SQL
- Pandas (ingestão de `.xlsx`)
- SQL

Fontes de dados

Arquivo	Formato	Registros
erp_pedidos_cabecalho_2025.csv	CSV	403
erp_pedidos_itens_2025.csv	CSV	995
crm_clientes_export.xlsx	Excel	183
comercial_canais.xlsx	Excel	8
vendedores.csv	CSV	42
legado_regioes_pipe.txt	TXT	8
cadastro_produtos_api_dump.json	JSON	72
logistica_entregas.json	JSON	322
atendimento_ocorrencias.ndjson	NDJSON	270

Fluxoporcamada

Bronze

Responsável por:

- ingestão raw
- schema explícito
- colunas técnicas
- rastreabilidade
- persistência em Delta

Todas as tabelas Bronze possuem:

- `source_file`
- `ingestion_date`
- `processing_timestamp`

Nenhuma regra de negócio é aplicada nesta camada.

Silver

Responsável por:

- limpeza
- parse de datas
- normalização
- deduplicação
- tratamento de nulos
- enriquecimento operacional
- quality logging

Principais tratamentos aplicados:

- múltiplos formatos de data (`coalesce`)
- casing inconsistente
- normalização PT/EN
- deduplicação determinística
- isolamento de registros órfãos
- recomputação de medidas inconsistentes

A Silver preserva rastreabilidade e evita descarte silencioso.

Gold

Responsável por:

- modelagem dimensional
- surrogate keys
- camada factual
- métricas executivas
- semantic layer

Modelo dimensional

Fato

```
gold.fato_pedidos
```

Granularidade:

```
1 linha por item de pedido válido
```

Volume final:

```
912 registros
```

Dimensões

- dim_cliente
- dim_produto
- dim_canal
- dim_regiao
- dim_vendedor
- dim_data

Métricas agregadas

- metricas_por_periodo

- metricas_por_regiao
- metricas_por_canal
- metricas_por_categoria
- metricas_operacionais

Semantic Layer

```
gold.vw_pedidos_analytics
```

View analítica pronta para consumo por BI.

A view:

- resolve dimensões
- expõe nomes de negócio
- filtra pedidos cancelados
- centraliza regras analíticas

Principais problemas encontrados

Fonte	Problema	Tratamento
pedidos_cabecalho	`status_order` nulo	`DESCONHECIDO`
pedidos_cabecalho	3 formatos de data	`coalesce(to_date(...))`
pedidos_itens	`item_status` nulo	`NAO_INFORMADO`
pedidos_itens	`total_item` divergente	recomputado
entregas	status PT/EN misturados	mapeamento canônico
clientes	estados misturados	normalização IBGE
vendedores	`regional_code = sul`	normalizado para `S`
regiões	código `XX` inválido	descartado
ocorrências	severity/event_type nulos	`NAO_INFORMADO`

Documentação detalhada:

```
docs/qualidade_dados.md
```

Decisões técnicas

Surrogate keys via sha2

As dimensões utilizam hash determinístico (`sha2`) para garantir:

- idempotência
- estabilidade em reprocessamentos
- independência de sequência de carga

Split Silver pedidos_cabecalho / pedidos_itens

O join entre cabeçalho e itens foi realizado apenas na Gold.

Isso preserva:

- granularidade
- separação de responsabilidade
- rastreabilidade

---

receita_liquida_item

O desconto existe no nível do pedido.

Para evitar dupla contagem em agregações por item:

- o desconto foi rateado proporcionalmente
- a receita líquida foi calculada por linha

---

dim_data dinâmica

A dimensão calendário é construída dinamicamente a partir:

```
MIN(data) – 30 dias
MAX(data) + 30 dias
```

Sem hardcode de período.

Replay handling e deduplicação

Durante a construção da Gold foi identificado fan-out causado por replay na Silver.

Causa raiz:

```
reprocessamento da silver_pedidos_cabecalho sem deduplicação determinística
```

Tratamento aplicado:

- deduplicação via `row_number()`
- partição por `order_id`
- manutenção do registro mais recente
- reconstrução completa da Gold

Resultado:

Resultado:

- granularidade preservada
- fato sem duplicidade
- pipeline idempotente

Validações executadas

- unicidade de surrogate keys
- integridade referencial fato → dimensões
- granularidade (`order_id + item_seq`)
- detecção de replay/fan-out
- cobertura da `dim_data`
- validação de medidas negativas
- status normalizados
- duplicidade em tabelas Silver
- consistência de métricas operacionais

As validações estão em:

```
tests/validacoes_pos_carga.py
tests/data_quality_checks.sql
```

Estratégia de reprocessamento

Bronze

`overwrite` completo.

A fonte do case é imutável e pequena.

Silver

`overwrite` completo.

A camada depende exclusivamente da Bronze.

Gold

`overwrite` completo.

Em ambiente produtivo:

- MERGE incremental
- janela móvel
- watermark

seriam mais adequados.

Premissas adotadas

- pedidos com status nulo não são considerados cancelados
- um pedido possui no máximo uma entrega válida
- região `SE` duplicada foi consolidada
- código `XX` foi descartado por não representar região válida

Limitações

- dimensões Type 1 (sem SCD)
- execução manual dos notebooks
- `.xlsx` via pandas
- ausência de orquestração
- ausência de framework formal de testes automatizados

Apesar disso, o projeto possui validações PySpark e SQL implementadas nos notebooks e em `tests/`.

Melhorias futuras

- SCD Type 2
- ingestão incremental via MERGE
- Databricks Workflows
- Unity Catalog
- DLT Expectations
- monitoramento automatizado
- dashboards executivos

Execução

Pré-requisitos:

- Databricks Community Edition
- DBR 12+
- arquivos carregados em:

```
/FileStore/case/sources/
```

Ordem de execução:

```
00_setup

→ bronze
→ silver
→ gold
→ tests
```

Todos os notebooks são autocontidos.

Resultaddfinal

O projeto entrega:

- pipeline Lakehouse ponta a ponta
- rastreabilidade operacional
- modelo dimensional
- camada factual íntegra
- métricas executivas
- semantic layer pronta para BI
- validações de qualidade
- reprocessamento idempotente
- documentação técnica completa

Evidências Técnicas

As evidências de execução, troubleshooting,
quality checks e validações finais podem ser encontradas em:

docs/evidencias_tecnicas.pdf