




# Explorando nossa camada de armazenamento Gold

## Como funciona o Delta Lake?

1.  **Dados físicos** armazenados como arquivos **Parquet**
2.  **Metadados e logs** salvos na pasta `_delta_log/` (arquivos `.json` )
3.  Cada modificação gera um novo **arquivo de log** com:
  - Tipo da operação (ex: `CREATE TABLE` , `MERGE` )
  - Arquivos adicionados ou removidos ( `add` / `remove` )
  - Estatísticas dos dados (min/max/null)

O Delta mantém o histórico de transações como um banco de dados, mas em arquivos no Data Lake!

## Primeiro, vamos analisar o conteúdo do arquivo parquet

```
In [2]: import pandas as pd
import pyarrow.parquet as pq

# Caminho do arquivo Parquet
parquet_path = 'data/example/part-00000-e86aa848-f9ed-4ac7-ac4b-8e1ddb3f2cc0-c000.snappy.parquet'

# Lê o conteúdo do arquivo Parquet como DataFrame
table = pq.read_table(parquet_path)
df_parquet = table.to_pandas()

# Exibir as primeiras linhas
df_parquet.head()
```

```
Out [2]:
```

	customer_id	total_amount
0	1003	300.75
1	1002	575.60
2	1001	330.00

## O que é Parquet?

- **Formato de armazenamento colunar**
- Otimizado para **consulta analítica**
- Alta **compressão e performance**
- Formato **aberto e interoperável**
- Utilizado amplamente por ferramentas como Spark, Hive, Trino e Presto

## Em seguida, vamos analisar o conteúdo do diretório `_delta_log`

O primeiro arquivo JSON registrou a criação inicial da tabela

```
In [4]: import json

# Caminhos dos arquivos de log Delta
caminho_arquivo_0 = 'data/example/_delta_log/00000000000000000000.json'
caminho_arquivo_1 = 'data/example/_delta_log/00000000000000000001.json'

# Função para ler as linhas JSON do Delta log
def ler_delta_log(caminho):
    with open(caminho, 'r') as f:
        linhas = f.readlines()
        return [json.loads(linha) for linha in linhas]

# Leitura dos arquivos
log_0 = ler_delta_log(caminho_arquivo_0)
log_1 = ler_delta_log(caminho_arquivo_1)
```

```
# Exemplo de visualização
```

```
for entrada in log_0:  
    print(json.dumps(entrada, indent=2))
```

```
{  
  "commitInfo": {  
    "timestamp": 1753606082636,  
    "operation": "CREATE TABLE",  
    "operationParameters": {  
      "isManaged": "false",  
      "description": null,  
      "partitionBy": "[]",  
      "properties": "{}"  
    },  
    "isolationLevel": "Serializable",  
    "isBlindAppend": true,  
    "operationMetrics": {},  
    "engineInfo": "Apache-Spark/3.4.3 Delta-Lake/2.4.0",  
    "txnId": "10dc12d1-8868-42b6-ab77-3b9062614afd"  
  },  
  "protocol": {  
    "minReaderVersion": 1,  
    "minWriterVersion": 2  
  },  
  "metaData": {  
    "id": "61290b4f-1eb5-47bb-a6b6-e629d6b3b86e",  
    "format": {  
      "provider": "parquet",  
      "options": {}  
    },  
    "schemaString": "{\"type\":\"struct\",\"fields\":[]}",  
    "partitionColumns": [],  
    "configuration": {},  
    "createTime": 1753606082467  
  }  
}
```

O segundo arquivo JSON registrou a inserção de dados na tabela

```
In [8]: for entrada in log_1:  
        print(json.dumps(entrada, indent=2))
```

```

{
  "commitInfo": {
    "timestamp": 1753606984287,
    "operation": "CREATE OR REPLACE TABLE AS SELECT",
    "operationParameters": {
      "isManaged": "false",
      "description": null,
      "partitionBy": "[]",
      "properties": "{}"
    },
    "readVersion": 0,
    "isolationLevel": "Serializable",
    "isBlindAppend": false,
    "operationMetrics": {
      "numFiles": "1",
      "numOutputRows": "3",
      "numOutputBytes": "780"
    },
    "engineInfo": "Apache-Spark/3.4.3 Delta-Lake/2.4.0",
    "txnId": "019326b7-9172-40c9-8fad-fa3a8a8f9090"
  }
}
{
  "metadata": {
    "id": "61290b4f-1eb5-47bb-a6b6-e629d6b3b86e",
    "format": {
      "provider": "parquet",
      "options": {}
    },
    "schemaString": "{\"type\":\"struct\",\"fields\":[{\"name\":\"customer_id\",\"type\":\"string\",\"nullable\":true,\"metadata\":{}},{\"name\":\"total_amount\",\"type\":\"double\",\"nullable\":true,\"metadata\":{}}]}",
    "partitionColumns": [],
    "configuration": {},
    "createTime": 1753606082467
  }
}
{
  "add": {
    "path": "part-00000-e86aa848-f9ed-4ac7-ac4b-8e1ddb3f2cc0-c000.snappy.parquet",
    "partitionValues": {},
    "size": 780,

```

```
"modificationTime": 1753606980000,  
"dataChange": true,  
"stats": "{\n  \"numRecords\":3,\n  \"minValues\":{\n    \"customer_id\": \"1001\",\n    \"total_amount\":300.75\n  },\n  \"maxValues\":{\n    \"customer_id\": \"1003\",\n    \"total_amount\":575.6\n  },\n  \"nullCount\":{\n    \"customer_id\":0,\n    \"total_amount\":0\n  }\n}"
```

## 🧱 O que é Delta Lake?

- Camada de armazenamento transacional construída sobre o **formato Parquet**
- Permite funcionalidades que um Data Lake tradicional não oferece:

### ✅ Funcionalidades principais:

- 🔄 **Controle de versão (Time Travel)**
- 🔒 **Transações ACID** (Atomicidade, Consistência, Isolamento, Durabilidade)
- 📂 **Leitura incremental** para alto desempenho em pipelines
- ✏️ **Atualizações, exclusões e merges** nativos (sem reescrever tudo)
- 📋 **Auditoria e governança** com logs detalhados

Base tecnológica essencial para a arquitetura Lakehouse moderna

In [ ]: