UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMATICA
GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

Sandro Victor Rosevel de Santana

Título: Análise comparativa qualitativa de Ferramentas de Manipulação de Dados em Workflows de ETL

Recife

2025

Sandro Victor Rosevel de Santana

Título: Análise comparativa qualitativa de Ferramentas de Manipulação de Dados em Workflows de ETL

Trabalho apresentado ao Programa de Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Área de Concentração: Engenharia de dados

Orientador (a): Adriano Lorena Inácio de Oliveira (alio@cin.ufpe.br)

Recife

2025

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Santana, Sandro Victor Rosevel de.
Análise comparativa qualitativa de Ferramentas de Manipulação de Dados em Workflows de ETL / Sandro Victor Rosevel de Santana. - Recife, 2025.
44, tab.

Orientador(a): Adriano Lorena Inácio de Oliveira
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Ciências da Computação - Bacharelado, 2025.

1. ETL. 2. Engenharia de dados. 3. Star Schema. 4. Desempenho de Pipelines. 5. PySpark. 6. Pandas.. I. Oliveira, Adriano Lorena Inácio de. (Orientação). II. Título.

000 CDD (22.ed.)

Sandro Victor Roosevel de Santana

Análise comparativa qualitativa de Ferramentas de Manipulação de Dados em Workflows de ETL

Trabalho apresentado ao Programa de Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em: 08 / 08 / 2025

BANCA EXAMINADORA

_____

Prof. Dr. Adriano Lorena Inácio de Oliveira (Orientadora)

Universidade Federal de Pernambuco

_____

Prof. Dr. Robson do Nascimento Fidalgo (Examinador Interno)

Universidade Federal de Pernambuco

AGRADECIMENTOS

A conclusão deste trabalho representa não apenas o encerramento de uma etapa acadêmica, mas também a soma de inúmeros apoios, incentivos e companhias que tornaram essa trajetória possível. Agradeço, em primeiro lugar, à minha noiva Ana Luiza, pelo amor, paciência e companheirismo constantes. Sua presença foi essencial em todos os momentos, dos mais leves aos mais desafiadores, e seu apoio incondicional me deu força para seguir até aqui. Aos meus pais, Verônica e Sidney, deixo minha profunda gratidão por todo o esforço, dedicação e valores transmitidos ao longo da vida. Sem o suporte de vocês, nada disso teria sido possível. À minha irmã Shirley e ao meu cunhado João Felipe, agradeço pelo carinho, incentivo e por fazerem parte desta caminhada com tanto afeto. Agradeço ainda a todos os amigos que fiz durante a graduação, que foram fundamentais ao longo do curso, especialmente durante os períodos difíceis enfrentados na pandemia. Cada momento de troca, apoio e convivência contribuiu imensamente para a realização deste trabalho e para minha formação pessoal e profissional.

RESUMO

Na era da informação, a capacidade de transformar grandes volumes de dados em insights acionáveis é um diferencial estratégico para organizações de todos os setores. Os processos de ETL (Extract, Transform, Load) ocupam papel central na Engenharia de Dados, permitindo integrar informações de múltiplas fontes, aplicar regras de negócio e disponibilizar resultados em ambientes analíticos como data warehouses ou data lakes. Este trabalho apresenta uma análise comparativa entre quatro ferramentas amplamente utilizadas na construção de pipelines ETL, Pandas, PySpark, Polars e SQL, avaliadas a partir de critérios como desempenho, consumo de recursos, tempo de execução, capacidade de paralelização e adequação a diferentes cenários de volume e infraestrutura. A base de dados utilizada é o Brazilian E-Commerce Public Dataset da Olist, modelada segundo o Star Schema, composto por uma tabela fato de transações e tabelas de dimensão associadas, padrão amplamente adotado em ambientes de Business Intelligence (KIMBALL; ROSS, 2008). A arquitetura do pipeline foi estruturada em três camadas, Bronze, Silver e Gold, que representam estágios progressivos de refinamento e enriquecimento dos dados (GONZALEZ; XIN; TEAM, 2020). Os experimentos revelaram diferenças significativas entre as ferramentas: o Polars apresentou o melhor desempenho em termos de tempo de execução, o SQL obteve bom equilíbrio entre performance e simplicidade, o Pandas se destacou em cenários com menor volume de dados, e o PySpark mostrou maior potencial em ambientes distribuídos de alta escala. Esses resultados oferecem subsídios práticos para que profissionais e organizações selecionem a tecnologia mais adequada às suas necessidades, considerando restrições de tempo, custo e recursos computacionais disponíveis.

Palavras-chaves: ETL. Star Schema. Pandas. PySpark. Polars. SQL. Data Lake. Desempenho de Pipelines.

ABSTRACT

In the information age, the ability to transform large volumes of data into actionable insights is a strategic differentiator for organizations across all sectors. ETL (Extract, Transform, Load) processes play a central role in Data Engineering, enabling the integration of information from multiple sources, the application of business rules, and the delivery of results to analytical environments such as data warehouses or data lakes. This work presents a comparative analysis of four widely used tools for building ETL pipelines — Pandas, PySpark, Polars, and SQL — evaluated according to criteria such as performance, resource consumption, execution time, parallelization capabilities, and suitability for different volume and infrastructure scenarios. The dataset used is the Brazilian E-Commerce Public Dataset from Olist, modeled according to the Star Schema, consisting of a fact table of transactions and associated dimension tables, a pattern widely adopted in Business Intelligence environments (KIMBALL; ROSS, 2008). The pipeline architecture was structured into three layers — Bronze, Silver, and Gold — representing progressive stages of data refinement and enrichment (GONZALEZ; XIN; TEAM, 2020). The experiments revealed significant differences between the tools: Polars showed the best performance in terms of execution time, SQL achieved a good balance between performance and simplicity, Pandas stood out in scenarios with smaller data volumes, and PySpark demonstrated greater potential in distributed, large-scale environments. These results provide practical insights for professionals and organizations to select the most suitable technology for their needs, considering time, cost, and available computational resources.

Keywords: ETL. Star Schema. Pandas. PySpark. Polars. SQL. Data Lake. Pipeline Performance.

LISTA DE CÓDIGOS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Vivemos em uma era de explosão de dados, na qual organizações das mais diversas áreas precisam transformar grandes volumes de dados brutos em informações úteis para a tomada de decisão. Nesse contexto, os processos de ETL (Extract, Transform, Load) desempenham um papel central na Engenharia de Dados, sendo responsáveis por extrair dados de múltiplas fontes, transformá-los conforme as necessidades analíticas e carregá-los em repositórios estruturados, como data warehouses ou data lakes.

Com o avanço das tecnologias, surgiram diversas ferramentas e bibliotecas voltadas para a manipulação de dados dentro de pipelines ETL. Entre elas, destacam-se Pandas, PySpark, Polars e SQL, cada uma oferecendo abordagens diferentes em termos de desempenho, escalabilidade, expressividade e facilidade de uso. Pandas e Polars são altamente eficientes em ambientes locais e ideais para cargas de dados menores, enquanto PySpark foi projetado para processamento distribuído em larga escala. Por outro lado, SQL continua sendo uma linguagem padrão para transformações em bancos de dados relacionais, com ampla adoção em sistemas corporativos.

A definição de qual ferramenta utilizar em um projeto de dados está frequentemente atrelada a fatores como volume dos dados, requisitos de performance, infraestrutura disponível e curva de aprendizado da equipe. Neste trabalho, propõe-se a realização de um estudo comparativo entre essas ferramentas, avaliando sua performance em pipelines ETL equivalentes, estruturados com base no modelo dimensional clássico (fato e dimensões), com dados de e-commerce.

Para garantir um cenário realista e controlado, será utilizado o ambiente gratuito Databricks Community Edition, que oferece suporte ao Apache Spark e ao Delta Lake, permitindo simular pipelines reais com uso de arquivos em diferentes formatos (CSV, Parquet e Delta). O conjunto de dados selecionado, o "Brazilian E-Commerce Public Dataset"da Olist, será utilizado para simular uma tabela fato de grande volumetria (itens de pedidos) e tabelas dimensão com menor volume (clientes, produtos, vendedores, pedidos).

A introdução de camadas Bronze, Silver e Gold na modelagem do pipeline reforça o alinhamento com práticas modernas de arquitetura de dados, e a estruturação dos notebooks simulará o comportamento de ferramentas de orquestração como o Azure Data Factory. Ao final, espera-se obter uma análise fundamentada sobre a aplicabilidade e eficiência de cada

ferramenta frente aos desafios típicos da Engenharia de Dados.

1.1 MOTIVAÇÃO

O crescimento exponencial na geração de dados, impulsionado pela digitalização de processos e pela disseminação de tecnologias emergentes, tem imposto novos desafios à área de Engenharia de Dados. A necessidade de estruturar pipelines de dados eficientes, escaláveis e aderentes às demandas de negócio torna cada vez mais relevante a compreensão das ferramentas e abordagens disponíveis para processos de ETL (Extract, Transform, Load). Em meio à diversidade de bibliotecas e frameworks, como Pandas, PySpark, Polars e SQL, surge a demanda por estudos comparativos que forneçam subsídios técnicos para a tomada de decisão quanto à melhor escolha de ferramenta, considerando aspectos como desempenho, facilidade de implementação, compatibilidade com arquiteturas modernas (como a abordagem em camadas Bronze, Silver e Gold) e adequação a diferentes cenários de volumetria e infraestrutura. Este trabalho se propõe a atender essa demanda, contribuindo com uma análise prática e sistematizada que auxilie profissionais e pesquisadores na seleção de tecnologias para projetos de integração e tratamento de dados em ambientes analíticos.

1.2 OBJETIVOS

1.2.1 Objetivo geral

Realizar uma avaliação comparativa entre ferramentas de manipulação de dados utilizadas na construção de pipelines ETL, em particular Pandas, PySpark, Polars e SQL. A seleção dessas ferramentas reflete diferentes paradigmas e contextos de uso: Pandas, amplamente difundido no meio acadêmico e corporativo, é ideal para processamento local e rápido desenvolvimento de protótipos; PySpark, projetado para processamento distribuído em larga escala, é uma referência em cenários de Big Data; Polars, de arquitetura moderna e alta performance, otimiza a execução local com baixo consumo de recursos; e SQL, linguagem declarativa universalmente adotada, mantém relevância na integração com sistemas de bancos de dados e ferramentas analíticas.

A comparação será realizada com base em critérios como desempenho, escalabilidade, consumo de recursos e aplicabilidade em diferentes cenários de volume. Ao fornecer uma

análise prática e sistematizada dessas ferramentas, este trabalho busca apoiar profissionais e estudantes da área de dados na tomada de decisões mais assertivas, promovendo o uso eficiente de recursos computacionais, a qualificação de projetos analíticos e, consequentemente, contribuindo para o avanço da cultura de dados em organizações públicas e privadas, com impacto direto na geração de conhecimento, inovação e apoio à tomada de decisão baseada em evidências.

1.2.2 Objetivos específicos

1. Construir pipelines ETL equivalentes utilizando as ferramentas Pandas, PySpark, Polars e SQL, com base em operações típicas como joins, filtros, agregações e transformações.
2. Modelar os dados com base em uma estrutura dimensional, utilizando uma tabela fato de alta volumetria e múltiplas tabelas de dimensão com volumetria média e baixa.
3. Implementar camadas de dados Bronze, Silver e Gold, aplicando boas práticas de arquitetura em data lakes, com uso de formatos otimizados como Parquet e Delta.
4. Executar os pipelines em ambientes gratuitos, priorizando o uso do Databricks Community Edition e simulações com notebooks para representar orquestrações de pipeline.
5. Medir e comparar métricas de desempenho, como tempo de execução, uso de memória, complexidade de implementação, legibilidade do código e suporte a paralelização.
6. Analisar os resultados obtidos nos testes, destacando vantagens e limitações de cada ferramenta para diferentes cenários de uso na Engenharia de Dados.
7. Fornecer recomendações práticas para a escolha da ferramenta mais adequada conforme o perfil da aplicação, a infraestrutura disponível e os objetivos do projeto.

2 FUNDAMENTAÇÃO

2.1 ENGENHARIA DE DADOS E O PROCESSO DE ETL

A Engenharia de Dados constitui uma das áreas centrais dentro do ecossistema de ciência de dados e inteligência analítica, sendo responsável pelo desenvolvimento e manutenção das estruturas técnicas que viabilizam a coleta, a organização, o armazenamento e a disponibilização de grandes volumes de dados. Essa disciplina atua de forma transversal em projetos de tecnologia da informação e é essencial para garantir que os dados estejam acessíveis, confiáveis e em conformidade com os requisitos analíticos e operacionais das organizações (KLEPPMANN, 2017). No contexto da Engenharia de Dados, os processos de ETL, acrônimo para Extract, Transform, Load, representam uma das práticas mais consolidadas e estratégicas. O ETL refere-se ao conjunto de operações que visa extrair dados de diferentes fontes (estruturadas ou não estruturadas), transformá- los segundo regras de negócio específicas e, por fim, carregá-los em repositórios analíticos, como data warehouses e data lakes (KIMBALL; ROSS, 2013). Esses repositórios servem de base para análises estatísticas, modelos de machine learning e dashboards corporativos.

A etapa de extração diz respeito à obtenção de dados de fontes distintas, como bancos relacionais, APIs, arquivos CSV, sistemas legados ou plataformas de streaming. Já a transformação compreende um amplo conjunto de operações, que incluem limpeza de dados, padronização de formatos, junções entre tabelas, conversões de tipos, enriquecimento e derivação de novas variáveis. Finalmente, a carga dos dados ocorre quando os registros transformados são inseridos nos sistemas-alvo, geralmente otimizados para consultas analíticas de alto desempenho (INMON, 2005). Com o crescimento do volume, da variedade e da velocidade dos dados, fenômeno conhecido como os "3Vs do Big Data"(LANEY, 2001), o paradigma clássico de ETL passou por transformações significativas. Surgiram modelos como o ELT (Extract, Load, Transform), em que os dados são primeiro carregados em ambientes escaláveis e só então transformados, aproveitando o poder de processamento paralelo de engines modernas como o Apache Spark. Além disso, ferramentas de código aberto e bibliotecas especializadas passaram a competir com soluções tradicionais, permitindo que times pequenos implementem pipelines robustos com custos reduzidos e maior flexibilidade.

Nesse cenário, a escolha das ferramentas e frameworks utilizados em pipelines ETL passou a depender de múltiplos fatores, tais como a volumetria dos dados, o modelo de execução (local

vs distribuído), a compatibilidade com a infraestrutura existente, a facilidade de manutenção e a curva de aprendizado das equipes. Como resultado, o domínio técnico e conceitual sobre os fundamentos do ETL tornou-se indispensável para engenheiros de dados, analistas e cientistas que atuam na cadeia de valor da informação.

2.2 MODELAGEM DIMENSIONAL: ESQUEMA ESTRELA

A modelagem dimensional é uma abordagem metodológica amplamente utilizada no desenvolvimento de sistemas analíticos e repositórios de dados, especialmente em ambientes voltados à inteligência de negócios (Business Intelligence – BI). Seu principal objetivo é estruturar os dados de forma a facilitar consultas analíticas, sumarizações, comparações temporais e visualizações gerenciais. Diferentemente da modelagem transacional, que prioriza normalização e integridade referencial, a modelagem dimensional privilegia o desempenho e a simplicidade de acesso aos dados para fins decisórios (KIMBALL; ROSS, 2013).

O modelo estrela (Star Schema) é a estrutura mais clássica dentro da modelagem dimensional. Ele é composto por uma tabela fato central, que armazena eventos quantitativos de interesse, como vendas, transações ou medições, e por diversas tabelas de dimensão, que contêm atributos descritivos que qualificam e contextualizam essas medidas (como produto, cliente, tempo, região, etc.). A tabela fato possui chaves estrangeiras que se relacionam diretamente com as chaves primárias das tabelas de dimensão, formando uma topologia que lembra uma estrela, daí a origem do nome.

Uma das principais características do Star Schema é ter suas dimensões desnormalizadas, o que demanda menos joins proporcionando maior desempenho nas consultas analíticas. Isso ocorre porque todas as dimensões se conectam diretamente à tabela fato, evitando relacionamentos intermediários que exigiriam múltiplas junções em cascata.

A adoção do Star Schema neste trabalho justifica-se por sua adequação aos objetivos de uma análise comparativa entre ferramentas de ETL. Ao utilizar uma tabela fato com alta volumetria, representando, por exemplo, os itens de pedidos em um cenário de e-commerce, e múltiplas dimensões com volumetria variada (clientes, produtos, vendedores, datas), é possível reproduzir um cenário realista de exploração de dados, com complexidade compatível às rotinas corporativas de análise. Além disso, o modelo estrela é amplamente compatível com ferramentas modernas de processamento distribuído e formatos otimizados de armazenamento, como Parquet e Delta Lake, o que o torna ideal para experimentações com diferentes bibliotecas de

manipulação de dados, como Pandas, PySpark, Polars e SQL.

2.3 ARQUITETURA EM CAMADAS

A arquitetura em camadas é uma abordagem consolidada na Engenharia de Dados moderna, especialmente em ambientes baseados em data lakes, que visa organizar o fluxo de dados em níveis estruturados conforme seu grau de tratamento, confiabilidade e prontidão analítica. Essa estratégia surgiu como resposta à complexidade crescente dos ecossistemas de dados, que passaram a exigir maior rastreabilidade, modularidade e controle sobre os processos de transformação (LEE; DAMJI; DAS, 2021).

A estrutura mais difundida segue o modelo Bronze $\rightarrow$ Silver $\rightarrow$ Gold, no qual os dados evoluem progressivamente desde sua forma bruta até versões refinadas e otimizadas para consumo. Cada camada cumpre uma função específica no ciclo de vida do dado:

- Camada Bronze: armazena os dados tal como foram extraídos das fontes originais, sem qualquer tratamento. Esta camada atua como o repositório imutável de dados brutos, garantindo integridade histórica e possibilidade de auditoria completa. Geralmente, os dados são persistidos em formatos simples como CSV, JSON ou Parquet, respeitando a estrutura original de ingestão.

- Camada Silver: representa os dados que passaram por processos de tratamento, limpeza, padronização e validação. Nessa camada, são realizadas operações como remoção de duplicidades, preenchimento de valores nulos, conversão de tipos e filtragem de inconsistências. Onde tem como objetivo produzir uma versão confiável dos dados.

- Camada Gold: representa a etapa final no fluxo de processamento de dados em arquiteturas estruturadas no modelo Bronze–Silver–Gold. Nessa fase, os dados já passaram por processos de extração, limpeza, padronização e integração, estando prontos para consumo analítico. Trata-se de um conjunto de dados altamente confiável, estruturado e otimizado para consultas de alto desempenho, geralmente modelado em esquemas dimensionais ou agregações específicas para atender indicadores de negócio e análises predefinidas. Seu público-alvo inclui analistas de BI, cientistas de dados e tomadores de decisão, que utilizam essas informações para geração de insights, elaboração de relatórios e suporte à tomada de decisões baseada em evidências. Na prática, a camada

Gold consolida e enriquece as informações, transformando dados operacionais em ativos estratégicos para a organização.

A adoção da arquitetura em camadas, especialmente no modelo Bronze–Silver–Gold, oferece benefícios significativos para a governança, a escalabilidade e a qualidade dos dados. Ao segmentar o pipeline em estágios distintos, é possível garantir maior rastreabilidade e controle sobre cada transformação, permitindo auditorias mais precisas e facilitando a identificação e correção de inconsistências (DATABRICKS, 2023). Essa abordagem também promove a reutilização de dados intermediários, reduzindo o retrabalho e otimizando recursos computacionais, uma vez que cada camada pode ser consumida por diferentes processos sem a necessidade de reprocessamento integral (GORELIK, 2019). Além disso, a separação clara entre dados brutos, tratados e analíticos favorece a colaboração entre equipes técnicas e de negócio, assegurando que os consumidores finais recebam informações confiáveis e alinhadas às necessidades estratégicas da organização, contribuindo diretamente para uma tomada de decisão mais rápida e assertiva (INMON, 2005).

No contexto deste trabalho, a aplicação da arquitetura em camadas permite simular pipelines reais de mercado, avaliando a capacidade de cada ferramenta em lidar com transformações sucessivas e com a persistência de dados em diferentes formatos e níveis de estruturação. Além disso, garante a comparabilidade entre as abordagens, ao padronizar o fluxo de processamento e os critérios de avaliação.

2.4 FERRAMENTAS ANALISADAS

Com o avanço das tecnologias de processamento de dados, uma ampla variedade de ferramentas e bibliotecas passou a ser utilizada na construção de pipelines de ETL, cada uma com características específicas quanto ao desempenho, à escalabilidade, à facilidade de uso e à adequação a diferentes contextos de infraestrutura e volumetria. A escolha da ferramenta mais apropriada depende de múltiplos fatores, por exemplo, quando o perfil dos dados envolve tabelas de pequeno ou médio porte processadas localmente, bibliotecas como Pandas se mostram vantajosas por sua simplicidade e ampla comunidade de suporte. Em contrapartida, quando o objetivo do projeto exige processar dados massivos e distribuídos, como logs de aplicações ou registros de sensores em tempo real, o PySpark oferece escalabilidade e paralelização nativas, permitindo o processamento em clusters. Já o Polars, com sua engine em Rust e suporte a exe-

cução preguiçosa (lazy execution), é indicado para cenários que demandam alto desempenho local aliado a baixo consumo de memória, tornando-o eficiente para análises exploratórias de grandes datasets em máquinas únicas. Por fim, o SQL, amplamente difundido e independente de linguagem de programação, é especialmente útil quando o nível de familiaridade da equipe com linguagens declarativas é alto e há necessidade de trabalhar diretamente sobre bancos de dados já estruturados, aproveitando recursos de otimização do próprio SGBD e integrando-se facilmente a diferentes infraestruturas corporativas. Nesta seção, são abordadas quatro tecnologias amplamente empregadas em ambientes de Engenharia de Dados: Pandas, PySpark, Polars e SQL.

2.4.1 Pandas

Pandas é uma biblioteca open source da linguagem Python, voltada à análise e manipulação de dados tabulares em memória. Desenvolvida por Wes McKinney em 2008, tornou-se uma das ferramentas mais populares para análise de dados em ambientes locais, sobretudo em projetos acadêmicos, exploratórios ou de pequeno e médio porte (MCKINNEY, 2017). Sua estrutura principal é o DataFrame, que fornece uma interface poderosa para operações como agregações, junções, filtragens, ordenações e transformações de dados.

Entre suas principais vantagens estão a facilidade de uso, a ampla documentação, a curva de aprendizado acessível e a integração fluida com outras bibliotecas do ecossistema Python, como NumPy, Matplotlib e Scikit-learn. No entanto, por operar inteiramente em memória, Pandas não é indicada para cargas massivas de dados, pois apresenta limitações de escalabilidade e de uso de múltiplos núcleos de processamento.

2.4.2 Pyspark

PySpark é a interface em Python para o Apache Spark, uma engine de processamento distribuído projetada para lidar com grandes volumes de dados em clusters. Sua arquitetura baseia-se em operações resilientes distribuídas (Resilient Distributed Datasets – RDDs) e em DataFrames otimizados, permitindo executar tarefas de ETL com paralelismo, tolerância a falhas e gerenciamento automático de recursos (ZAHARIA et al., 2016).

Ao contrário de bibliotecas locais como Pandas, PySpark é capaz de escalar horizontalmente por meio do particionamento e distribuição das tarefas entre diferentes nós de um cluster.

Além disso, possui integração nativa com o Delta Lake, possibilitando persistência de dados em formatos otimizados, com suporte a transações ACID, versionamento e verificação de esquema (schema enforcement). Sua principal desvantagem é a maior complexidade de configuração e o tempo de resposta inicial mais elevado, especialmente em pequenas cargas de dados.

2.4.3 Polars

Polars é uma biblioteca de manipulação de dados que vem ganhando destaque por seu alto desempenho e arquitetura moderna. Escrita em Rust, uma linguagem de programação de baixo nível e alta performance, Polars oferece bindings para Python e executa operações de forma paralela e altamente eficiente (HEYDEN, 2021). Diferentemente do modelo tradicional de execução imediata (eager execution), Polars utiliza execução preguiçosa (lazy execution), o que permite ao otimizador reorganizar e combinar operações antes de executá-las, resultando em melhorias significativas de desempenho.

Além da velocidade, Polars apresenta consumo reduzido de memória e excelente desempenho com arquivos colunares, como Parquet. Por outro lado, seu ecossistema ainda está em expansão, com menor documentação e menor compatibilidade com bibliotecas consolidadas em Python. Apesar disso, vem sendo adotado progressivamente por profissionais que buscam maior performance em ambientes locais sem recorrer ao overhead de soluções distribuídas.

2.4.4 Structured Query Language (SQL)

Structured Query Language (SQL) é a linguagem padrão para gerenciamento e manipulação de dados em sistemas de bancos de dados relacionais. Desde sua padronização pela ANSI e ISO, SQL se consolidou como a base de operações de leitura e escrita em estruturas tabulares, estando presente em quase todos os SGBDs comerciais e open source (DATE, 2003).

SQL destaca-se por sua clareza sintática, baixo custo de implementação e ampla familiaridade entre profissionais de tecnologia. É particularmente eficaz para operações declarativas em bancos estruturados, como filtros, agregações, joins e subconsultas. Em plataformas modernas como o Apache Spark, é possível utilizar SQL em conjunto com engines de execução distribuída (Spark SQL), unindo a expressividade da linguagem com o poder de processamento paralelo. Sua principal limitação reside na restrição a modelos relacionais e na dificuldade de compor pipelines complexos com múltiplos estágios de transformação sem perda de legibilidade.

3 METODOLOGIA

Este trabalho adota uma abordagem experimental e comparativa para avaliar o desempenho e a aplicabilidade de diferentes ferramentas de manipulação de dados no contexto de pipelines de ETL. A metodologia foi estruturada com o objetivo de garantir reprodutibilidade, equidade nas condições de teste e relevância prática para cenários reais da Engenharia de Dados. As ferramentas analisadas foram: Pandas, PySpark, Polars e SQL. O ambiente de testes, os critérios de avaliação, o modelo de dados e a organização dos pipelines foram definidos conforme descrito a seguir.

3.1 AMBIENTE DE EXECUÇÃO

Todos os experimentos foram realizados na plataforma Databricks Community Edition, que oferece um ambiente unificado para desenvolvimento, execução e monitoramento de notebooks. A escolha dessa plataforma deve-se à sua compatibilidade com múltiplas linguagens (Python, SQL, Scala), ao suporte nativo ao Apache Spark e ao Delta Lake, bem como à padronização da infraestrutura de execução, o que elimina interferências decorrentes de variações de hardware ou sistema operacional local.

Dentro desse ambiente:

- Pandas e Polars foram executados diretamente via notebooks Python, com arquivos hospedados no Databricks File System (DBFS);
- PySpark foi utilizado por meio da SparkSession nativa da plataforma;
- SQL foi implementado tanto por meio de notebooks SQL;

3.2 CONJUNTO DE DADOS

Foi utilizado o Brazilian E-Commerce Public Dataset da empresa Olist, amplamente adotado em benchmarks acadêmicos e empresariais por representar um cenário realista de comércio eletrônico. O conjunto de dados inclui informações sobre pedidos, clientes, produtos, vendedores e avaliações. Sua volumetria que varia entre 2mb e 100mb e heterogeneidade de atributos

o tornam adequado para a simulação de um modelo dimensional clássico, com uma tabela fato de alta cardinalidade e múltiplas tabelas de dimensão.

3.3 MODELAGEM DIMENSIONAL

Os dados foram modelados com base no esquema estrela (Star Schema), uma técnica amplamente adotada em ambientes analíticos. A modelagem foi composta por:

- Tabela fato: representando os itens de pedidos, com elevada volumetria.
- Tabelas dimensão: representando entidades como clientes, produtos, vendedores e datas.

Essa estrutura permite a aplicação de operações típicas de ETL, como joins, filtros, transformações, agregações e formatações.

3.4 ARQUITETURA EM CAMADAS

Os pipelines foram organizados segundo a arquitetura em camadas Bronze $\rightarrow$ Silver $\rightarrow$ Gold, conforme descrito abaixo:

- Camada Bronze: ingestão dos dados brutos em arquivos CSV, sem qualquer tratamento.
- Camada Silver: aplicação de transformações, limpeza, validação e normalização dos dados.
- Camada Gold: integração dos dados tratados em uma estrutura analítica consolidada, com persistência em formatos otimizados (Parquet ou Delta).

A utilização dessa arquitetura permitiu avaliar a capacidade de cada ferramenta em realizar transformações progressivas e persistir dados em formatos analíticos.

3.5 ORGANIZAÇÃO DOS PIPELINES

Os pipelines foram implementados por meio de notebooks modulares no Databricks, simulando práticas de orquestração adotadas em ferramentas como o Azure Data Factory. Cada notebook representou uma etapa do fluxo (ingestão, transformação, carga), permitindo a execução sequencial ou isolada. Essa abordagem visou refletir condições próximas a um ambiente

de produção, com controle de etapas, reusabilidade e rastreabilidade. Os dados lidos da camada bronze foram comuns a todas as ferramentas, porém os dados gerados para a camada silver e gold foram separados e identificados pela ferramenta que o gerou.

3.6 CRITÉRIOS DE AVALIAÇÃO

As ferramentas foram avaliadas a partir de métricas quantitativas e qualitativas, descritas a seguir:

Tabela 1 – Critérios de avaliação

Métrica	Descrição	Ferramenta de Medição
Tempo de execução	Duração total da execução do pipeline	time, time.time(), Spark UI
Paralelização	Capacidade de execução distribuída ou multithread	Observação de execução em cluster
Consumo de memória	avaliação do uso de memória durante a execução	Compute/Cluster metrics

3.7 PERSISTÊNCIA DE DADOS

Para garantir uma análise mais justa e equilibrada entre as ferramentas avaliadas, todos os dados da camada Bronze foram disponibilizados em formato CSV, comum a todas as execuções, servindo como ponto de partida padronizado. A partir desse estágio, as camadas Silver e Gold foram persistidas exclusivamente no formato Parquet para todas as ferramentas, assegurando uniformidade no formato final e aproveitando suas vantagens, como compressão eficiente, leitura seletiva de colunas e ampla compatibilidade. Foram criados diretórios separados para a escrita das camadas Silver e Gold de acordo com a ferramenta utilizada, permitindo a execução isolada de cada tecnologia sem interferência nos resultados. Essa abordagem preservou a comparabilidade entre os cenários testados, ao mesmo tempo em que respeitou as particularidades e fluxos de processamento de cada solução.

4 IMPLEMENTAÇÃO DAS PIPELINES

Este capítulo descreve a construção prática dos pipelines de ETL utilizando as quatro ferramentas selecionadas para análise comparativa: Pandas, PySpark, Polars e SQL. O desenvolvimento seguiu os princípios metodológicos definidos no capítulo anterior, buscando garantir isonomia entre as abordagens e fidelidade à arquitetura em camadas Bronze, Silver e Gold.

A implementação foi realizada na plataforma Databricks Community Edition, que permitiu a integração das diferentes tecnologias em um ambiente de execução unificado, com suporte a leitura e escrita em diversos formatos, execução de notebooks e persistência em sistemas de arquivos distribuídos. A escolha desse ambiente visou minimizar interferências externas, garantindo que os resultados obtidos refletissem unicamente o comportamento das ferramentas testadas frente às mesmas transformações e estrutura de dados.

Todas as implementações foram baseadas no mesmo conjunto de dados, extraído do repositório público da Olist. As transformações realizadas foram cuidadosamente replicadas em cada ferramenta, com o objetivo de assegurar comparabilidade nas medições de desempenho, legibilidade e compatibilidade com a arquitetura proposta.

4.1 ESTRUTURA GERAL DOS PIPELINES

Para manter a uniformidade entre os experimentos e permitir uma análise comparativa consistente, foi definida uma estrutura padrão de pipeline, composta por três fases principais, que correspondem às camadas Bronze, Silver e Gold do modelo arquitetônico adotado.

A estrutura geral do pipeline consistiu na leitura dos dados da camada Bronze, comuns a todas as ferramentas, seguida da aplicação dos tratamentos necessários de acordo com as particularidades de cada conjunto de dados. Os resultados dessas transformações foram então salvos na camada Silver no formato Parquet. Na etapa seguinte, as tabelas foram lidas da camada Silver e integradas por meio de operações de join, resultando na construção da camada Gold, também armazenada no formato Parquet.

- Camada Bronze – Ingestão de Dados Brutos: Os dados foram carregados diretamente dos arquivos CSV disponibilizados no dataset da Olist. Nenhuma transformação foi aplicada neste estágio, que visa apenas armazenar os dados na forma original em um repositório unificado.

- Camada Silver – Tratamento e Preparação dos Dados: Nesta etapa, foram aplicadas transformações essenciais como:
  - Conversão de tipos de dados (strings para datetime, inteiros etc.);
  - Remoção de valores nulos e registros inconsistentes;
  - Padronização de nomes de colunas;
  - Filtros específicos (como exclusão de pedidos cancelados).
- Camada Gold – Integração e Consolidação para Análise: Os dados tratados foram integrados por meio de junções entre a tabela fato (order_items) e as tabelas dimensão (orders, customers, products, sellers). Foram também derivadas colunas adicionais (como valor total por item), e os dados finais foram persistidos em formato otimizado (Parquet ou Delta, a depender da ferramenta). Esta camada representa a base analítica final, apta para consultas e visualizações.

4.2 IMPLEMENTAÇÃO COM PANDAS

A implementação dos pipelines utilizando a biblioteca Pandas foi organizada de forma modular, com cada notebook representando uma etapa específica do fluxo de ETL. A separação por tabelas e camadas permitiu maior clareza no rastreamento das transformações e simulou práticas adotadas em pipelines orquestrados por ferramentas como Azure Data Factory ou Apache Airflow.

A seguir, detalha-se a estrutura adotada e as transformações realizadas em cada notebook.

4.2.1 Camada Silver(Pandas)

Notebook 1 - Ingestão e Tratamento: customers_bronze_to_silver Este notebook foi responsável por:

- Leitura do arquivo customers.csv em formato CSV;
- Conversão de colunas para tipos apropriados (por exemplo, customer_zip_code_prefix para string);
- Renomeação de colunas para nomenclatura padronizada (customer_id $\rightarrow$ id_cliente);

- Remoção de registros com campos nulos críticos.

Código Fonte 1 – Ingestão e Tratamento: customers_bronze_to_silver

```
import pandas as pd

customers = pd.read_csv('/dbfs/FileStore/tables/olist_customers_dataset.csv')

# Renomear e padronizar
customers = customers.rename(columns={
    'customer_id': 'id_cliente',
    'customer_unique_id': 'id_cliente_unico',
    'customer_zip_code_prefix': 'cep',
    'customer_city': 'cidade',
    'customer_state': 'estado'
})

# Drop de registros nulos
customers = customers.dropna()

# Escrita camada prata
customers.to_parquet('/dbfs/FileStore/tables/silver/customers.parquet', index=
    False)
```

Fonte: Elaborado pelo autor (2025)

Notebook 2 - Ingestão e Tratamento: products_bronze_to_silver Transformações aplicadas:

- Leitura do arquivo products.csv;
- Conversão de tipos e remoção de colunas irrelevantes para análise;
- Tratamento de nulos em colunas como product_category_name e product_weight_g.

Código Fonte 2 – Ingestão e Tratamento: products_bronze_to_silver

```
products = pd.read_csv('/dbfs/FileStore/tables/olist_products_dataset.csv')

products = products.rename(columns={
    'product_id': 'id_produto',
    'product_category_name': 'categoria',
    'product_weight_g': 'peso',
    'product_length_cm': 'comprimento',
    'product_height_cm': 'altura',
    'product_width_cm': 'largura'
})
```

```
11
products = products.dropna(subset=['categoria', 'peso'])
13
products.to_parquet('/dbfs/FileStore/tables/silver/products.parquet', index=False
    )
```

Fonte: Elaborado pelo autor (2025)

Notebook 3 - Ingestão e Tratamento: sellers_bronz_to_silver Neste notebook, os dados dos vendedores foram apenas padronizados e verificados quanto à completude.

Código Fonte 3 – Ingestão e Tratamento: sellers_bronz_to_silver

```
sellers = pd.read_csv('/dbfs/FileStore/tables/olist_sellers_dataset.csv')
2
sellers = sellers.rename(columns={
4    'seller_id': 'id_vendedor',
    'seller_zip_code_prefix': 'cep',
6    'seller_city': 'cidade',
    'seller_state': 'estado'
8 })

10 sellers = sellers.dropna()

12 sellers.to_parquet('/dbfs/FileStore/tables/silver/sellers.parquet', index=False)
```

Fonte: Elaborado pelo autor (2025)

Notebook 4 - Ingestão e Tratamento: orders_bronze_to_silver Aqui foram feitas transformações nas colunas de datas e o filtro de cancelamentos.

Código Fonte 4 – Ingestão e Tratamento: orders_bronze_to_silver

```
orders = pd.read_csv('/dbfs/FileStore/tables/olist_orders_dataset.csv')
2
orders = orders.rename(columns={
4    'order_id': 'id_pedido',
    'customer_id': 'id_cliente',
6    'order_status': 'status',
    'order_purchase_timestamp': 'data_compra'
8 })

10 orders['data_compra'] = pd.to_datetime(orders['data_compra'])

12 # Filtrar apenas pedidos aprovados
orders = orders[orders['status'] != 'canceled']
14
orders.to_parquet('/dbfs/FileStore/tables/silver/orders.parquet', index=False)
```

Fonte: Elaborado pelo autor (2025)

Notebook 5 - Ingestão e Tratamento: order_items_bronze_to_silver Esta etapa tratou a tabela fato, realizando ajustes nos campos monetários e conversões de tipos.

Código Fonte 5 – Ingestão e Tratamento: order_items_bronze_to_silver

```
items = pd.read_csv('/dbfs/FileStore/tables/olist_order_items_dataset.csv')

items = items.rename(columns={
    'order_id': 'id_pedido',
    'order_item_id': 'id_item',
    'product_id': 'id_produto',
    'seller_id': 'id_vendedor',
    'price': 'preco',
    'freight_value': 'frete'
})

items = items.dropna()

items.to_parquet('/dbfs/FileStore/tables/silver/order_items.parquet', index=False
    )
```

Fonte: Elaborado pelo autor (2025)

4.2.2 Construção da Camada Gold

Notebook 6 - Construção da Camada Gold: build_gold_fato_pandas

Este notebook leu todos os arquivos da camada prata e executou as junções entre fato e dimensões.

Código Fonte 6 – Construção da Camada Gold: build_gold_fato_pandas

```
# Leitura camada prata
items = pd.read_parquet('/dbfs/FileStore/tables/silver/order_items.parquet')
orders = pd.read_parquet('/dbfs/FileStore/tables/silver/orders.parquet')
products = pd.read_parquet('/dbfs/FileStore/tables/silver/products.parquet')
sellers = pd.read_parquet('/dbfs/FileStore/tables/silver/sellers.parquet')
customers = pd.read_parquet('/dbfs/FileStore/tables/silver/customers.parquet')

# Jun  es
df = items.merge(orders, on='id_pedido', how='inner')
df = df.merge(products, on='id_produto', how='left')
df = df.merge(sellers, on='id_vendedor', how='left')
df = df.merge(customers, on='id_cliente', how='left')

# Valor total
df['valor_total'] = df['preco'] + df['frete']
```

```
16
# Persist ncia camada gold
18 df.to_parquet('/dbfs/FileStore/tables/gold/fato_pandas.parquet', index=False)
```

Fonte: Elaborado pelo autor (2025)

4.3 IMPLEMENTAÇÃO COM PYSPARK

A implementação dos pipelines com PySpark seguiu a mesma estrutura modular aplicada à biblioteca Pandas, com notebooks independentes para cada etapa de transformação. O uso do PySpark permitiu aproveitar a execução distribuída nativa da engine Spark, favorecendo o processamento de grandes volumes de dados de forma escalável e eficiente.

Todas as leituras e gravações foram feitas no formato otimizado Delta Lake, com suporte a transações ACID, versionamento e verificação de esquema. A seguir, são descritas as principais transformações realizadas em cada notebook do pipeline com PySpark.

4.3.1 Camada Silver(PySpark)

Notebook 1 - Ingestão e Tratamento: customers_bronze_to_silver

Código Fonte 7 – Ingestão e Tratamento: customers_bronze_to_silver

```
customers = spark.read.option("header", True).csv("dbfs:/FileStore/tables/
    olist_customers_dataset.csv")
2
customers = customers.selectExpr(
4     "customer_id as id_cliente",
      "customer_unique_id as id_cliente_unico",
6     "cast(customer_zip_code_prefix as string) as cep",
      "customer_city as cidade",
8     "customer_state as estado"
).dropna()
10
customers.write.format("delta").mode("overwrite").save("dbfs:/FileStore/tables/
    silver/customers")
```

Fonte: Elaborado pelo autor (2025)

Notebook 2 - Ingestão e Tratamento: products_bronze_to_silver

Código Fonte 8 – Ingestão e Tratamento: products_bronze_to_silver

```
1 products = spark.read.option("header", True).csv("dbfs:/FileStore/tables/
    olist_products_dataset.csv")
```

```
products = products.selectExpr(
    "product_id as id_produto",
    "product_category_name as categoria",
    "cast(product_weight_g as int) as peso",
    "cast(product_length_cm as int) as comprimento",
    "cast(product_height_cm as int) as altura",
    "cast(product_width_cm as int) as largura"
).dropna(subset=["categoria", "peso"])

products.write.format("delta").mode("overwrite").save("dbfs:/FileStore/tables/silver/products")
```

Fonte: Elaborado pelo autor (2025)

Notebook 3 - Ingestão e Tratamento: sellers_bronze_to_silver

Código Fonte 9 – Ingestão e Tratamento: sellers_bronze_to_silver

```
sellers = spark.read.option("header", True).csv("dbfs:/FileStore/tables/olist_sellers_dataset.csv")

sellers = sellers.selectExpr(
    "seller_id as id_vendedor",
    "cast(seller_zip_code_prefix as string) as cep",
    "seller_city as cidade",
    "seller_state as estado"
).dropna()

sellers.write.format("delta").mode("overwrite").save("dbfs:/FileStore/tables/silver/sellers")
```

Fonte: Elaborado pelo autor (2025)

Notebook 4 - Ingestão e Tratamento: orders_bronze_to_silver

Código Fonte 10 – Ingestão e Tratamento: orders_bronze_to_silver

```
orders = spark.read.option("header", True).csv("dbfs:/FileStore/tables/olist_orders_dataset.csv")

orders = orders.selectExpr(
    "order_id as id_pedido",
    "customer_id as id_cliente",
    "order_status as status",
    "cast(order_purchase_timestamp as timestamp) as data_compra"
).filter("status != 'canceled'")

orders.write.format("delta").mode("overwrite").save("dbfs:/FileStore/tables/silver/orders")
```

Fonte: Elaborado pelo autor (2025)

Notebook 5 - Ingestão e Tratamento: order_items_bronze_to_silver

Código Fonte 11 – Ingestão e Tratamento: order_items_bronze_to_silver

```
items = spark.read.option("header", True).csv("dbfs:/FileStore/tables/
    olist_order_items_dataset.csv")

items = items.selectExpr(
    "order_id as id_pedido",
    "order_item_id as id_item",
    "product_id as id_produto",
    "seller_id as id_vendedor",
    "cast(price as double) as preco",
    "cast(freight_value as double) as frete"
).dropna()

items.write.format("delta").mode("overwrite").save("dbfs:/FileStore/tables/silver
    /order_items")
```

Fonte: Elaborado pelo autor (2025)

4.3.2 Construção da Camada Gold

Notebook 6 - Construção da Camada Gold: build_gold_fato_pyspark

Código Fonte 12 – Ingestão e Tratamento: build_gold_fato_pyspark

```
# Leitura camada prata
items = spark.read.format("delta").load("dbfs:/FileStore/tables/silver/
    order_items")
orders = spark.read.format("delta").load("dbfs:/FileStore/tables/silver/orders")
products = spark.read.format("delta").load("dbfs:/FileStore/tables/silver/
    products")
sellers = spark.read.format("delta").load("dbfs:/FileStore/tables/silver/sellers"
    )
customers = spark.read.format("delta").load("dbfs:/FileStore/tables/silver/
    customers")

# Jun  es
df = items.join(orders, "id_pedido", "inner") \
          .join(products, "id_produto", "left") \
          .join(sellers, "id_vendedor", "left") \
          .join(customers, "id_cliente", "left") \
          .withColumn("valor_total", items.preco + items.frete)

# Escrita camada gold
```

```
16 df.write.format("delta").mode("overwrite").save("dbfs:/FileStore/tables/gold/
       fato_pyspark")
```

Fonte: Elaborado pelo autor (2025)

4.4 IMPLEMENTAÇÃO COM POLARS

A implementação dos pipelines utilizando a biblioteca Polars seguiu o mesmo modelo modular das demais abordagens, com notebooks específicos para cada tabela de dimensão, a tabela fato e a construção da camada Gold. A abordagem lazy execution foi empregada para otimizar o plano de execução e reduzir o uso de recursos computacionais.

Todos os arquivos de saída foram persistidos no formato Parquet, dada a compatibilidade do Polars com arquivos colunarizados e a ausência de suporte direto ao Delta Lake.

4.4.1 Camada Silver(Polars)

Notebook 1 - Ingestão e Tratamento: customers_bronze_to_silver

Código Fonte 13 – Ingestão e Tratamento: customers_bronze_to_silver

```
import polars as pl

customers = pl.read_csv("/Volumes/workspace/benchmark/bronze/
    olist_customers_dataset.csv").lazy()

customers = customers.rename({
    "customer_id": "id_cliente",
    "customer_unique_id": "id_cliente_unico",
    "customer_zip_code_prefix": "cep",
    "customer_city": "cidade",
    "customer_state": "estado"
}).drop_nulls()

customers.collect().write_parquet("/Volumes/workspace/benchmark/polars/silver/
    customers.parquet")
```

Fonte: Elaborado pelo autor (2025)

Notebook 2 -Ingestão e Tratamento: products_bronze_to_silver

Código Fonte 14 – Ingestão e Tratamento: products_bronze_to_silver

```
products = pl.read_csv("/Volumes/workspace/benchmark/bronze/
    olist_products_dataset.csv").lazy()
```

```
products = products.rename({
    "product_id": "id_produto",
    "product_category_name": "categoria",
    "product_weight_g": "peso",
    "product_length_cm": "comprimento",
    "product_height_cm": "altura",
    "product_width_cm": "largura"
}).drop_nulls(["categoria", "peso"])

products.collect().write_parquet("/Volumes/workspace/benchmark/polars/silver/
    products.parquet")
```

Fonte: Elaborado pelo autor (2025)

Notebook 3 - Ingestão e Tratamento: sellers_bronze_to_silver

Código Fonte 15 – Ingestão e Tratamento: sellers_bronze_to_silver

```
sellers = pl.read_csv("/Volumes/workspace/benchmark/bronze/olist_sellers_dataset.
    csv").lazy()

sellers = sellers.rename({
    "seller_id": "id_vendedor",
    "seller_zip_code_prefix": "cep",
    "seller_city": "cidade",
    "seller_state": "estado"
}).drop_nulls()

sellers.collect().write_parquet("/Volumes/workspace/benchmark/polars/silver/
    sellers.parquet")
```

Fonte: Elaborado pelo autor (2025)

Notebook 4 -Ingestão e Tratamento: orders_bronze_to_silver

Código Fonte 16 – Ingestão e Tratamento: orders_bronze_to_silver

```
orders = pl.read_csv("/Volumes/workspace/benchmark/bronze/olist_orders_dataset.
    csv").lazy()

orders = orders.rename({
    "order_id": "id_pedido",
    "customer_id": "id_cliente",
    "order_status": "status",
    "order_purchase_timestamp": "data_compra"
}).filter(pl.col("status") != "canceled")

orders = orders.with_columns([
    pl.col("data_compra").str.strptime(pl.Datetime, "%Y-%m-%d %H:%M:%S")
])
```

```
orders.collect().write_parquet("/Volumes/workspace/benchmark/polars/silver/orders.parquet")
```

Fonte: Elaborado pelo autor (2025)

Notebook 5 -Ingestão e Tratamento: order_items_bronze_to_silver

Código Fonte 17 – Ingestão e Tratamento: order_items_bronze_to_silver

```
items = pl.read_csv("/Volumes/workspace/benchmark/bronze/olist_order_items_dataset.csv").lazy()

items = items.rename({
    "order_id": "id_pedido",
    "order_item_id": "id_item",
    "product_id": "id_produto",
    "seller_id": "id_vendedor",
    "price": "preco",
    "freight_value": "frete"
}).drop_nulls()

items.collect().write_parquet("/Volumes/workspace/benchmark/polars/silver/order_items.parquet")
```

Fonte: Elaborado pelo autor (2025)

4.4.2 Construção da Camada Gold

Notebook 6 - Construção da Camada Gold: build_gold_fato_polars

Código Fonte 18 – Construção da Camada Gold: build_gold_fato_polars

```
# Leitura camada prata
items = pl.read_parquet("/Volumes/workspace/benchmark/polars/silver/order_items.parquet").lazy()
orders = pl.read_parquet("/Volumes/workspace/benchmark/polars/silver/orders.parquet").lazy()
products = pl.read_parquet("/Volumes/workspace/benchmark/polars/silver/products.parquet").lazy()
sellers = pl.read_parquet("/Volumes/workspace/benchmark/polars/silver/sellers.parquet").lazy()
customers = pl.read_parquet("/Volumes/workspace/benchmark/polars/silver/customers.parquet").lazy()

# Jun  es e coluna derivada
df = (items
      .join(orders, on="id_pedido", how="inner")
```

```
    .join(products, on="id_produto", how="left")
    .join(sellers, on="id_vendedor", how="left")
    .join(customers, on="id_cliente", how="left")
    .with_columns([
        (pl.col("preco") + pl.col("frete")).alias("valor_total")
    ])
)

df.collect().write_parquet("/Volumes/workspace/benchmark/polars/gold/fato_polars.parquet")
```

Fonte: Elaborado pelo autor (2025)

4.5 IMPLEMENTAÇÃO COM SQL (SPARK SQL)

A implementação com SQL foi realizada diretamente em notebooks do Databricks, utilizando comandos declarativos para leitura, transformação e junção dos dados. Cada notebook representa uma etapa do pipeline, e as tabelas intermediárias e finais foram salvas utilizando o formato Delta Lake, explorando os recursos de transações ACID e otimizações nativas do Spark.

Todos os datasets foram lidos da camada Bronze, e as saídas intermediárias foram armazenadas na Silver zone. A camada Gold foi construída a partir da junção das tabelas tratadas com persistência em formato Delta.

4.5.1 Camada Silver(SQL)

Notebook 1 - Ingestão e Tratamento: customers_bronze_to_silver

Código Fonte 19 – Ingestão e Tratamento

```
CREATE OR REPLACE TEMP VIEW bronze_customers AS
SELECT * FROM csv.`/Volumes/workspace/benchmark/bronze/olist_customers_dataset.csv`;

CREATE OR REPLACE TABLE delta.`/Volumes/workspace/benchmark/sql/silver/customers`
    AS
SELECT
    customer_id AS id_cliente,
    customer_unique_id AS id_cliente_unico,
    CAST(customer_zip_code_prefix AS STRING) AS cep,
    customer_city AS cidade,
    customer_state AS estado
```

```
FROM bronze_customers
WHERE customer_id IS NOT NULL;
```

Fonte: Elaborado pelo autor (2025)

Notebook 2 - Ingestão e Tratamento: products_bronze_to_silver

Código Fonte 20 – Ingestão e Tratamento

```
CREATE OR REPLACE TEMP VIEW bronze_products AS
SELECT * FROM csv.`/Volumes/workspace/benchmark/bronze/olist_products_dataset.csv`;

CREATE OR REPLACE TABLE delta.`/Volumes/workspace/benchmark/sql/silver/products` AS
SELECT
    product_id AS id_produto,
    product_category_name AS categoria,
    CAST(product_weight_g AS INT) AS peso,
    CAST(product_length_cm AS INT) AS comprimento,
    CAST(product_height_cm AS INT) AS altura,
    CAST(product_width_cm AS INT) AS largura
FROM bronze_products
WHERE product_category_name IS NOT NULL AND product_weight_g IS NOT NULL;
```

Fonte: Elaborado pelo autor (2025)

Notebook 3 - Ingestão e Tratamento: sellers_bronze_to_silver

Código Fonte 21 – Ingestão e Tratamento

```
CREATE OR REPLACE TEMP VIEW bronze_sellers AS
SELECT * FROM csv.`/Volumes/workspace/benchmark/bronze/olist_sellers_dataset.csv`;

CREATE OR REPLACE TABLE delta.`/Volumes/workspace/benchmark/sql/silver/sellers` AS
SELECT
    seller_id AS id_vendedor,
    CAST(seller_zip_code_prefix AS STRING) AS cep,
    seller_city AS cidade,
    seller_state AS estado
FROM bronze_sellers
WHERE seller_id IS NOT NULL;
```

Fonte: Elaborado pelo autor (2025)

Notebook 4 - Ingestão e Tratamento: orders_bronze_to_silver

Código Fonte 22 – Ingestão e Tratamento

```
CREATE OR REPLACE TEMP VIEW bronze_orders AS
```

```
SELECT * FROM csv.`/Volumes/workspace/benchmark/bronze/olist_orders_dataset.csv`;

CREATE OR REPLACE TABLE delta.`/Volumes/workspace/benchmark/sql/silver/orders` AS
SELECT
    order_id AS id_pedido,
    customer_id AS id_cliente,
    order_status AS status,
    CAST(order_purchase_timestamp AS TIMESTAMP) AS data_compra
FROM bronze_orders
WHERE order_status != 'canceled';
```

Fonte: Elaborado pelo autor (2025)

Notebook 5 - Ingestão e Tratamento: order_items_bronze_to_silver

Código Fonte 23 – Ingestão e Tratamento

```
CREATE OR REPLACE TEMP VIEW bronze_order_items AS
SELECT * FROM csv.`/Volumes/workspace/benchmark/bronze/olist_order_items_dataset.
    csv`;

CREATE OR REPLACE TABLE delta.`/Volumes/workspace/benchmark/sql/silver/
    order_items` AS
SELECT
    order_id AS id_pedido,
    order_item_id AS id_item,
    product_id AS id_produto,
    seller_id AS id_vendedor,
    CAST(price AS DOUBLE) AS preco,
    CAST(freight_value AS DOUBLE) AS frete
FROM bronze_order_items
WHERE order_id IS NOT NULL;
```

Fonte: Elaborado pelo autor (2025)

4.5.2 Construção da Camada Gold

Notebook 6 - Construção da Camada Gold: build_gold_fato_sql

Código Fonte 24 – Ingestão e Tratamento

```
-- Leitura das tabelas silver como views tempor rias
CREATE OR REPLACE TEMP VIEW order_items AS
SELECT * FROM delta.`/Volumes/workspace/benchmark/sql/silver/order_items`;

CREATE OR REPLACE TEMP VIEW orders AS
SELECT * FROM delta.`/Volumes/workspace/benchmark/sql/silver/orders`;

```

```
CREATE OR REPLACE TEMP VIEW products AS
SELECT * FROM delta.`/Volumes/workspace/benchmark/sql/silver/products`;

CREATE OR REPLACE TEMP VIEW sellers AS
SELECT * FROM delta.`/Volumes/workspace/benchmark/sql/silver/sellers`;

CREATE OR REPLACE TEMP VIEW customers AS
SELECT * FROM delta.`/Volumes/workspace/benchmark/sql/silver/customers`;

-- Constru  o da tabela fato com jun  es e c lculo
CREATE OR REPLACE TABLE delta.`/Volumes/workspace/benchmark/sql/gold/fato_sql` AS
SELECT
    i.*,
    o.data_compra,
    p.categoria,
    s.cidade AS cidade_vendedor,
    c.estado AS estado_cliente,
    (i.preco + i.frete) AS valor_total
FROM order_items i
INNER JOIN orders o ON i.id_pedido = o.id_pedido
LEFT JOIN products p ON i.id_produto = p.id_produto
LEFT JOIN sellers s ON i.id_vendedor = s.id_vendedor
LEFT JOIN customers c ON o.id_cliente = c.id_cliente;
```

Fonte: Elaborado pelo autor (2025)

5 ANÁLISE DOS RESULTADOS

Este capítulo apresenta a análise dos resultados obtidos a partir da execução dos pipelines de ETL desenvolvidos com as ferramentas Pandas, PySpark, Polars e SQL. A avaliação comparativa foi realizada com base nos critérios previamente estabelecidos na metodologia, a saber: tempo de execução, uso de memória, facilidade de implementação, legibilidade do código, suporte à paralelização e compatibilidade com a arquitetura em camadas Bronze, Silver e Gold.

As medições foram conduzidas em ambiente controlado, utilizando a plataforma Databricks Community Edition, com os mesmos dados de entrada e transformações equivalentes em cada ferramenta. Para reduzir a influência de eventuais variações de desempenho causadas por flutuações do ambiente ou latências momentâneas, cada pipeline foi executado oito vezes, e o resultado apresentado corresponde à média aritmética dos tempos de execução obtidos. A seguir, os resultados são apresentados de forma comparativa.

5.1 TEMPO DE EXECUÇÃO

O tempo de execução dos pipelines é um dos principais critérios de avaliação no contexto de Engenharia de Dados, especialmente em ambientes com restrições de tempo de processamento ou alto volume de dados. Neste estudo, foram cronometradas todas as etapas da transformação dos dados da camada Bronze para a Silver (B2S) e da Silver para a Gold (S2G) em cada ferramenta. Os resultados obtidos demonstram variações significativas de desempenho entre as abordagens.

A biblioteca Polars apresentou os menores tempos em praticamente todas as etapas. Na leitura e transformação da tabela customers, por exemplo, a execução foi concluída em apenas 1 segundo, enquanto o tempo de junção e construção da camada Gold (orderskpiS2G) levou 13 segundos. Sua arquitetura baseada em execução preguiçosa (lazy execution) e uso eficiente de CPU explicam esse desempenho superior, mesmo em ambiente local e sem cluster distribuído.

A linguagem SQL, executada via Spark SQL no Databricks, também obteve resultados expressivos. Os tempos de execução variaram entre 5 e 12 segundos nas etapas de Bronze para Silver, e 15 segundos na etapa final de junção. Por ser uma linguagem declarativa e integrada ao engine do Spark, o SQL conseguiu otimizar os planos de execução internamente,

com bom balanceamento entre performance e simplicidade.

O Pandas apresentou desempenho razoável nas etapas com menor volume de dados, como customers e itens, com tempos entre 2 e 3 segundos. No entanto, nas tabelas com maior volume ou maior complexidade de transformação, como orders, sellers e a construção da Gold, os tempos aumentaram para 18 a 23 segundos. A limitação da execução sequencial e o uso intensivo de memória RAM são fatores que impactam seu desempenho à medida que a carga de dados cresce.

O PySpark, apesar de ser projetado para grandes volumes e execução distribuída, teve tempos relativamente mais altos para as transformações testadas. O tempo para tratar a tabela customers foi de 35 segundos, enquanto a transformação da tabela sellers foi a mais demorada, com 1 minuto e 15 segundos. A etapa de construção da Gold (orderskpiS2G) levou 40 segundos. O custo de inicialização da engine Spark, a sobrecarga de gerenciamento de sessões e o volume moderado de dados explicam o tempo superior das execuções, o que reforça que o Spark se mostra mais vantajoso em contextos com cargas massivas e operações altamente distribuídas.

Tabela 2 – A Tabela sintetiza a média dos tempos de execução observados para cada ferramenta

Etapa	**Pandas**	**PySpark**	**Polars**	**SQL**
Customers (B2S)	5s	66s	1s	12s
Itens (B2S)	4s	66s	3s	5s
Orders (B2S)	29s	31s	4s	9s
Products (B2S)	18s	67s	3s	6s
Sellers (B2S)	23s	68s	4s	8s
Orders KPI (S2G)	23s	65s	13s	15s

5.2 USO DE MEMÓRIA

O uso de memória é um fator crítico na escolha de ferramentas de manipulação de dados, especialmente em ambientes com recursos computacionais limitados ou em processos que exigem execução simultânea de múltiplos pipelines. Embora os testes tenham sido executados em um ambiente unificado (Databricks Community Edition), as ferramentas apresentaram comportamentos distintos quanto à alocação e gerenciamento de memória durante as transformações.

A biblioteca Pandas, por operar inteiramente em memória (RAM), apresentou os maiores picos de consumo entre as ferramentas testadas. Cada DataFrame carregado mantém-se integralmente na memória durante toda a execução do pipeline, o que pode comprometer a estabilidade em máquinas com baixa capacidade de RAM. Essa característica, embora ofereça agilidade em acessos e transformações simples, torna a biblioteca sensível ao aumento do volume de dados ou à complexidade das operações, como múltiplos joins e agregações.

Por outro lado, Polars demonstrou excelente eficiência no uso de memória. Sua arquitetura baseada em vetores colunarizados, somada à execução preguiçosa (lazy evaluation), permite ao otimizador reorganizar o plano de execução antes de alocar recursos. Isso reduz significativamente a ocupação de memória, tornando Polars altamente indicado para ambientes locais que demandam performance sem sobrecarregar a infraestrutura.

O comportamento de PySpark se destaca por seu modelo de execução distribuída. Mesmo sendo executado em ambiente monousuário (como no Databricks Community Edition), o Spark gerencia a memória de forma particionada e segmentada por tarefas. Isso proporciona maior controle sobre a escalabilidade, mas também adiciona overhead de gerenciamento. Em termos práticos, PySpark apresentou consumo de memória moderado, adaptando-se bem a operações em larga escala, embora menos eficiente que Polars em operações simples.

A linguagem SQL, quando utilizada via Spark SQL, apresentou consumo de memória equilibrado. Como os planos de execução SQL são otimizados pelo próprio Spark Catalyst, a engine é capaz de calcular previamente a melhor estratégia de execução, minimizando carregamentos desnecessários em memória. Em geral, o uso de memória foi similar ao observado em PySpark, porém com maior previsibilidade e menor sobrecarga em transformações diretas.

5.3 CAPACIDADE DE PARALELIZAÇÃO

A capacidade de paralelização é um critério crucial no processamento de dados em larga escala, pois está diretamente associada à eficiência computacional e ao tempo total de execução. Ferramentas que aproveitam múltiplos núcleos ou ambientes distribuídos são mais indicadas para pipelines de alto volume e cargas intensivas, mas gostaria de destacar que a paralelização nesse estudo foi realizada com o uso de workflows do databricks, onde pudemos chamar e executar notebooks paralelamente. Nesta subseção, avalia-se como cada ferramenta explora o paralelismo, tanto em ambientes locais quanto em arquiteturas distribuídas.

A biblioteca Pandas, por padrão, executa suas operações de forma sequencial, utilizando

apenas um núcleo da CPU. Isso significa que, independentemente da máquina ou ambiente em que é executado, o desempenho está limitado ao processamento linear dos dados. Embora existam bibliotecas complementares que introduzem paralelismo (como Dask), elas não fazem parte do escopo nativo do Pandas. Portanto, em termos de paralelização, a ferramenta apresenta limitações importantes, especialmente em cenários com alta volumetria.

Polars, por outro lado, é uma das ferramentas mais otimizadas para paralelismo em ambientes locais. Sua engine é escrita em Rust e foi projetada para aproveitar múltiplos núcleos de CPU de forma eficiente. Mesmo sem exigir configuração explícita de paralelismo, o Polars executa operações vetorizadas e paralelas, tanto em execução preguiçosa (lazy) quanto imediata (eager). Isso permite ganhos de performance notáveis em pipelines complexos mesmo em máquinas comuns, sem a necessidade de uma infraestrutura distribuída.

PySpark é a ferramenta mais completa no que se refere à paralelização. Sua arquitetura é nativamente distribuída, baseada em DAGs (Grafos Acíclicos Dirigidos), e opera por meio da divisão dos dados em partições e do envio de tarefas para múltiplos workers. Em clusters Spark completos, é possível escalar horizontalmente para centenas de nós, com tolerância a falhas e balanceamento de carga. Mesmo em ambientes locais, como no Databricks Community Edition, o PySpark simula essa divisão de tarefas, o que traz benefícios mesmo que limitados pela infraestrutura de execução. A principal vantagem do PySpark é sua robustez em ambientes empresariais de Big Data.

A linguagem SQL, executada sobre o Spark SQL, também se beneficia da engine distribuída do Spark. A interface Catalyst realiza otimizações no plano lógico e físico de execução, aproveitando paralelismo de forma transparente ao usuário. Dessa forma, mesmo scripts SQL simples podem ser executados de forma paralela quando o ambiente Spark está corretamente configurado. O usuário não precisa se preocupar com particionamento manual ou definição de jobs — a execução paralela é automaticamente inferida pelo engine.

6 CONCLUSÃO

Este trabalho teve como objetivo avaliar comparativamente quatro ferramentas de manipulação de dados aplicadas a pipelines de ETL, Pandas, PySpark, Polars e SQL, com base em critérios como desempenho, consumo de recursos, legibilidade do código, capacidade de paralelização, facilidade de implementação e compatibilidade com a arquitetura em camadas (Bronze, Silver e Gold).

Através da construção de pipelines equivalentes em ambiente controlado (Databricks Community Edition), foi possível observar com clareza os pontos fortes e limitações de cada abordagem. As medições foram feitas com base em dados reais de e-commerce, utilizando um modelo dimensional com tabela fato e múltiplas tabelas de dimensão, o que proporcionou um cenário realista e representativo da prática em Engenharia de Dados.

Dentre os principais achados, destacam-se:

Polars foi a ferramenta com melhor desempenho em termos de tempo de execução e uso de memória, demonstrando-se altamente eficiente em ambientes locais, mesmo sem suporte a execução distribuída.

SQL via Spark apresentou ótimo equilíbrio entre performance, legibilidade e compatibilidade com arquiteturas modernas, sendo ideal para profissionais que preferem uma linguagem declarativa e acessível.

PySpark foi a ferramenta mais robusta em termos de escalabilidade e paralelização, adequada para grandes volumes de dados e ambientes de produção distribuídos. No entanto, apresentou maior complexidade de implementação e maior overhead em operações menores.

Pandas manteve sua posição como uma ferramenta acessível e produtiva para pequenos e médios volumes de dados, embora tenha demonstrado limitações claras em desempenho e escalabilidade para pipelines mais complexos.

Todas as ferramentas foram capazes de estruturar os dados segundo a arquitetura em camadas proposta. Contudo, somente PySpark e SQL ofereceram suporte nativo ao Delta Lake, permitindo controle de versões, consistência transacional e integração com catálogos analíticos, o que as torna mais indicadas para ambientes corporativos com altos requisitos de governança e rastreabilidade.

Do ponto de vista prático, este trabalho oferece subsídios importantes para profissionais e equipes que atuam na modelagem de pipelines de dados. A escolha da ferramenta ideal

deve considerar não apenas o volume de dados e a infraestrutura disponível, mas também a maturidade técnica da equipe, os objetivos do projeto e os requisitos de manutenção e escalabilidade.

Como trabalho futuro, sugere-se a ampliação do estudo para outros formatos de dados (como JSON, Avro ou dados em tempo real), a inclusão de bibliotecas paralelas como Dask ou Modin no escopo local, e a avaliação de custo computacional em ambientes pagos e escaláveis. Também seria relevante testar a integração das ferramentas com orquestradores como Apache Airflow ou Azure Data Factory, aproximando ainda mais o estudo de cenários reais de produção.

REFERÊNCIAS

DATABRICKS. *Medallion Architecture: Organizing Data in the Lakehouse*. 2023. <https://docs.databricks.com/en/lakehouse/medallion.html>. Acesso em: 11 ago. 2025.

DATE, C. J. *An Introduction to Database Systems*. 8th. ed. [S.l.]: Addison-Wesley, 2003. ISBN 978-0321197849.

GONZALEZ, J.; XIN, R.; TEAM, D. *Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores*. 2020. <https://databricks.com/blog/2020/04/22/delta-lake-high-performance-acid-table-storage-over-cloud-object-stores.html>. Whitepaper.

GORELIK, A. *The Enterprise Big Data Lake: Delivering the Promise of Big Data and Data Science*. [S.l.]: O'Reilly Media, 2019. ISBN 978-1491931554.

HEYDEN, R. V. van der. *Polars: Fast DataFrames in Rust and Python*. 2021. <https://www.pola.rs/>. Acessado em 2025.

INMON, W. H. *Building the Data Warehouse*. 4th. ed. [S.l.]: Wiley, 2005. ISBN 978-0764599446.

KIMBALL, R.; ROSS, M. *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*. [S.l.]: John Wiley & Sons, 2008.

KIMBALL, R.; ROSS, M. *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*. 3rd. ed. [S.l.]: Wiley, 2013. ISBN 978-1118530801.

KLEPPMANN, M. *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. [S.l.]: O'Reilly Media, 2017. ISBN 978-1449373320.

LANEY, D. *3D Data Management: Controlling Data Volume, Velocity and Variety*. 2001. <https://blogs.gartner.com/doug-laney/>. Meta Group Note, posteriormente incorporado ao conceito de Big Data.

LEE, D.; DAMJI, J.; DAS, T. *Data Engineering with Databricks*. 2021. <https://docs.databricks.com/>. Acessado em 2025.

MCKINNEY, W. *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. 2nd. ed. [S.l.]: O'Reilly Media, 2017. ISBN 978-1491957660.

ZAHARIA, M.; XIN, R.; WENDELL, P.; DAS, T.; ARMBRUST, M.; DAVE, A.; MENG, X.; ROSEN, J.; VENKATARAMAN, S.; STOICA, I.; FRANKLIN, M.; GHODSI, A.; SHENKER, S. Apache spark: A unified engine for big data processing. In: *Communications of the ACM*. [S.l.]: ACM, 2016. v. 59, n. 11, p. 56–65.