
**Universidade Federal de Pernambuco**

Departamento de Engenharia da Computação

Curso de Engenharia da Computação

# Avaliando a performance entre Apache Iceberg e Apache Hive utilizando esquema estrela

Trabalho de Conclusão de Curso de Graduação

por

Enrique Laborão Monteiro

Orientador: Prof. Robson Fidalgo

Recife, Abril / 2025

Enrique Laborão Monteiro

# Avaliando a performance entre Apache Iceberg e Apache Hive utilizando esquema estrela

Monografia apresentada ao Curso de Engenharia da Computação, como requisito parcial para a obtenção do Título de Bacharel em Engenharia da Computação, Centro de Informática da Universidade Federal de Pernambuco.

Orientador: Prof. Robson Fidalgo

Recife

2025

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

Monteiro, Enrique Laborão.
Avaliando a performance entre Apache Iceberg e Apache Hive utilizando esquema estrela / Enrique Laborão Monteiro. - Recife, 2025.
53 p : il., tab.

Orientador(a): Robson do Nascimento Fidalgo
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Engenharia da Computação - Bacharelado, 2025.
Inclui referências.

1. Data Lakehouse. 2. Apache Iceberg. 3. Esquema Estrela. 4. Star Schema Benchmark. I. Fidalgo, Robson do Nascimento. (Orientação). II. Título.

000 CDD (22.ed.)

Enrique Laborão Monteiro

**Avaliando a performance entre Apache Iceberg e Apache Hive utilizando esquema estrela**

Tese de graduação apresentada, como requisito parcial para obtenção do título de Bacharel em Engenharia da Computação do Centro de Informatica da Universidade Federal de Pernambuco. Área de concentração: Banco de dados.

Aprovado em: 08 de Abril de 2025

Banca Examinadora:

---

Prof. Dr. Robson Fidalgo (Orientador)
Centro de Informática - UFPE

---

Prof. Dr. Eduardo Tavares
Centro de Informática - UFPE

Recife

2025

## Agradecimentos

*Eu gostaria de agradecer a todos que me ajudaram durante esta jornada, especialmente a:*

*Minha família, pelo apoio e pela base que me foi dada.*

*Ao professor Robson Fidalgo pela orientação e acompanhamento no desenvolvimento da tese.*

*Professores Marinho, José Fernandes, Fabiano Nader e Vieira Filho, por toda mentoria e ensinamentos.*

*Iandé Coutinho pela compreensão e por prover o ambiente para realização dos experimentos.*

*Marcello Pontes e Raniel Silva pelo direcionamento técnico e ajuda nas revisões.*

*Finalmente, eu gostaria de agradecer a todos os amigos do Team Grande pelo companheirismo dos últimos anos.*

*Um vencedor nunca abandona a luta,*
*e aquele que a abandona jamais vencerá.*

Napoleon Hill

**RESUMO**

Com a crescente necessidade de sistemas de processamento e armazenamento de dados eficientes e escaláveis, os Data Lakes (DL) tornaram-se uma das arquiteturas mais populares. Assim, desafios relacionados à sua organização, consistência e desempenho de consultas se tornaram tópicos relevantes. A introdução dos Data Lakehouses (DLHs) visa resolver esses problemas, trazendo uma mescla entre atributos de DL e Data Warehouse (DW). O Apache Iceberg é uma solução emergente que visa abordar esses problemas, oferecendo uma camada refinada de gerenciamento de metadados, com suporte a transações ACID e otimizações para a leitura de grandes conjuntos de dados. Por outro lado, tabelas Hive, uma tecnologia mais consolidada, ainda são amplamente utilizadas para estruturar dados tabulares em Data Lakes. O objetivo desta investigação é avaliar o desempenho de tabelas gerenciadas pelo Apache Iceberg e pelo Hive em um cenário analítico real utilizando o Star Schema Benchmark. Além disso, busca-se analisar o impacto da inserção de dados, verificando como a fragmentação de arquivos e sua otimização influenciam o desempenho de cada solução. A pesquisa também visa identificar as vantagens potenciais do Apache Iceberg em relação ao Apache Hive, contribuindo para a melhores decisões tecnológicas para ambientes de Big Data.

Palavras-chave: Data Lakehouse, Apache Iceberg, Apache Hive, Esquema Estrela, Star Schema Benchmark

**ABSTRACT**

With the increasing demand for efficient and scalable data processing and storage, the Data Lake (DL) has become one of the most popular architectures. As such, problems with it's management, consistency and query performance have become relevant topics. The introduction of Data Lakehouses (DLHs) aims to solve these issues, bringing a mix of Data Lake and Data Warehouse (DW) features. Apache Iceberg is an emerging solution that seeks to fix these issues, providing a refined metadata management layer, with support for ACID transactions and optimizations for querying large datasets. On the other side, Hive tables, a more consolidated technology, are still widely used to structure tabular data in Data Lakes. The objective of this current thesis is to evaluate the analytical performance of tables managed by both Apache Iceberg and Hive in a real-world scenario using the Star Schema Benchmark. Additionally, it's investigated the impact of data insertions and how the fragmentation of the data files and it's optimization influences the performance of each tool. The research aims to identify potential advantages of using Apache Iceberg in relation to Hive, contributing to better informed technological decisions in big data applications.

Keywords: Data Lakehouse, Apache Iceberg, Apache Hive, AWS Athena, Star Schema Benchmark

## LISTA DE FIGURAS

## LISTA DE TABELAS

## LISTA DE SIGLAS

| | |
|---|---|
| ACID | Atomicidade, Consistência, Isolamento, Durabilidade |
| DL | Data Lake |
| DLH | Data Lakehouse |
| DW | Data Warehouse |
| ETL | Extract Transform Lado |
| OS | Object Storage |
| PR | Performance Relativa |
| SF10 | Scale Factor 10 |
| SF100 | Scale Factor 100 |
| SGBD | Sistema Gerenciador de Banco de Dados |
| SSB | Star Schema Benchmark |
| TPC | Transaction Processing Performance Council |

**SUMÁRIO**

# 1 INTRODUÇÃO

## 1.1 Contexto

A crescente necessidade de sistemas de armazenamento e processamento de dados eficientes e escaláveis impulsionou a criação de diferentes soluções para o gerenciamento e a consulta de grandes volumes de dados. Nesse cenário, os Data Lakes (DL) tornaram-se uma das arquiteturas preferidas, oferecendo um método versátil e escalável para lidar com informações diversas. No entanto, desafios relacionados à organização, consistência e desempenho de consultas em DLs continuam sendo relevantes. A introdução dos Data Lakehouses (DLHs) visa resolver esses problemas, trazendo uma mescla entre atributos de Data Lake e Data Warehouse (DW) [1].

O Apache Iceberg é uma solução emergente que visa abordar esses problemas oferecendo uma camada refinada de gerenciamento de metadados [2]. Ele fornece recursos como suporte a transações ACID (Atomicidade, Consistência, Isolamento, Durabilidade), particionamento dinâmico e otimizações para a leitura de grandes conjuntos de dados. Por outro lado, o Apache Hive [3], uma tecnologia mais consolidada, ainda é amplamente utilizado para estruturar dados tabulares em Data Lakes.

## 1.2 Motivação

A motivação para este trabalho reside na necessidade de compreender como soluções modernas para gerenciamento de tabelas em Data Lakes podem influenciar a eficiência e a escalabilidade de sistemas de Big Data. Embora o Apache Hive já esteja estabelecido no mercado, o Apache Iceberg apresenta uma alternativa que pretende solucionar desafios importantes, como escalabilidade e consistência transacional. Essa nova alternativa vem, porém, com os desafios de um ecossistema ainda em processo de maturação, mais camadas de abstração para configurar e manter, e desafios de otimização que ainda devem ser melhor explorados.

## 1.3 **Objetivos**

O objetivo desta investigação é avaliar o desempenho de tabelas gerenciadas pelo Apache Iceberg e pelo Apache Hive em um cenário analítico real utilizando o Star Schema Benchmark [4]. Além disso, busca-se analisar o impacto da inserção de dados, verificando como a fragmentação de arquivos e sua otimização influenciam o desempenho de cada solução. A pesquisa também visa identificar as vantagens potenciais do Apache Iceberg em relação ao Apache Hive, contribuindo para a melhores decisões tecnológicas para ambientes de Big Data.

## 1.4 **Organização do documento**

O projeto é organizado da seguinte forma: O Capítulo 2 dá um referencial teórico, introduzindo conceitos relevantes à investigação; o Capítulo 3 sumariza os trabalhos relacionados com esta tese; o Capítulo 4 define a metodologia utilizada nos experimentos; o capítulo 5 apresenta os resultados dos experimentos. Por fim, o capítulo 6 traz as conclusões e trabalhos futuros.

# 2 FUNDAMENTAÇÃO TEÓRICA

## 2.1 Big Data

Na revolução digital, sobretudo com o uso de dispositivos móveis mais difundido na sociedade, a maneira como as empresas coletam, armazenam e analisam dados mudou bastante. Com o crescimento exponencial das fontes de dados, como redes sociais, sensores IoT, transações financeiras e atividades online, o volume de informações disponíveis atingiu proporções sem precedentes. Esse fenômeno deu origem ao conceito de Big Data, que se refere a conjuntos de informações tão vastos e complexos que os métodos tradicionais de processamento se tornam ineficazes [5].

O Big Data não é apenas sobre o volume de dados, mas também sobre a variedade (diferentes tipos), a velocidade (a rapidez com que os registros são gerados e precisam ser processados). Alguns autores também acrescentam a veracidade e o valor como componentes do Big Data [6] [7]. Gerenciar e extrair valor dessas informações é um desafio significativo para as organizações, que buscam *insights* para impulsionar decisões estratégicas, melhorar a eficiência operacional e criar novas oportunidades de negócios.

## 2.2 Ecossistema Hadoop

Para atender às crescentes demandas de armazenar e processar esses dados, foi desenvolvida uma família de soluções complementares conhecida como Ecossistema Hadoop [8] [9]. Ele foi inicialmente estruturado em cima de 2 componentes: o Hadoop Distributed File System (HDFS), que permite o armazenamento de grandes volumes de dados de forma distribuída, e o MapReduce, que permite o processamento paralelo de grandes massas de dados. Essas tecnologias possibilitaram a escalabilidade dos sistemas com hardware comum, reduzindo os custos e facilitando o gerenciamento e análise de volumes de dados massivos. Várias outras ferramentas foram integradas ao ecossistema, visando novas funcionalidades e respondendo às necessidades das organizações por um gerenciamento facilitado e performático de suas informações.

## 2.3 **Cloud Computing**

A introdução da computação em nuvem também transformou o cenário de Big Data [10]. As empresas agora tinham à sua disposição uma infraestrutura escalável, com implantação facilitada e custos por utilização. Isso diminuiu massivamente os custos para implementação de grandes sistemas e possibilitou arquiteturas mais robustas e performáticas. Outra vantagem dos sistemas em nuvem são serviços gerenciados como Amazon EMR [11], Databricks [12] e Google Dataproc [13], que simplificam o gerenciamento das ferramentas, exigindo menos expertise técnica para a adoção dessas soluções.

### 2.3.1 Object Storage

Uma das ferramentas mais impactantes da computação em nuvem foi a introdução do *Object Storage* (OS) [14]. Ao invés de gerenciar armazenamento massivo através do HDFS, tornou-se possível utilizar serviços nativos de provedores de cloud para fazer essa função com performance, durabilidade e escalabilidade superiores, e custos substancialmente menores. Essa mudança possibilitou capacidades de armazenamento quase ilimitadas a custos razoáveis e de fácil acesso, auxiliando a popularização dos Data Lakes (abordados na seção 2.5). Entre as principais soluções de *Object Storage* estão: Amazon S3 [15], Google Cloud Storage [16] e Azure Blob Storage [17]

## 2.4 **Data Warehouse**

O conceito de Data Warehouse (DW) surge ainda nos anos 80 [18], visando projetar uma arquitetura de referência que auxilie instituições no armazenamento e manipulação dos seus dados para a tomada de decisão. O termo foi refinado e popularizado por Inmon, onde foi definido como: "Um data warehouse é uma coleção de dados orientada a assuntos, integrada, não volátil e variante no tempo que apoia as decisões da administração." [19]

Atualmente, o Warehouse é entendido como um repositório organizado de dados em formato tabular, que integra vários sistemas e visa cargas de trabalho analíticas. Sua implementação é usualmente feita com DBMSs tradicionais ou soluções específicas para *warehousing* como Redshift [20] ou BigQuery [21].

## 2.5 **Data lake**

O Data Lake (DL) surgiu como um repositório central de dados implementado em cima de armazenamento distribuído e projetado para lidar com altos volumes de dados vindos de diferentes fontes [22]. Porém, diferentemente de um DW, que segue uma estrutura tabular rígida e previamente definida, os DLs permitem processar dados em seus formatos de origem, sejam eles estruturados, semiestruturados ou não estruturados. Essa abordagem oferece maior flexibilidade para processamento e integração.

Contudo, o DL também tem limitações. Por ser mais flexível com o esquema dos dados, os DLs sofrem com problemas de governança, gerenciamento de metadados e controle de qualidade. Também por serem construídos em cima de um sistema de arquivos, eles não possuem suporte para transações, uma ferramenta crucial para integração simultânea de vários sistemas.

## 2.6 **Hive**

Uma das principais barreiras do Hadoop MapReduce era sua interface em Java, o que criava uma barreira para muitos engenheiros. Pensando em facilitar a gestão de metadados e facilitar o acesso, foi criado o Apache Hive em 2010 [3] [23]. Hive é uma solução de warehousing para Hadoop que provê uma interface de consulta SQL-like chamada Hive-QL, assim como o Hive Metastore (HMS), que serve como catálogo de dados. O Metastore, além de schemas e definição de tabelas, também provê estatísticas que auxiliam no planejamento da execução de queries.

O usuário pode inserir dados de fontes externas em diversos formatos por meio de uma *Data Manipulation Language* (DML). Essas informações podem ser então consultadas por meio da Hive-QL utilizando projeções, junções, agregações, uniões e subconsultas. Uma das principais vantagens do Hive é a possibilidade de criar tabelas a partir de arquivos que já estão armazenados no *Object Store*. Essas *external tables* possibilitam integrar dados de qualquer ferramenta, desde que estejam em um formato de arquivo reconhecido pelo Hive.

O Hive, porém, considera a pasta, e não arquivos específicos, para a definição das tabelas, o que impõe desafios para a performance e transações ACID.

Apesar da engine de consulta do Hive ter sido amplamente substituída por soluções mais modernas e completas como SparkSQL, Trino e Presto, a implementação das tabelas Hive e o uso do Hive Metastore ainda são predominantes em muitas organizações.

## 2.7 **Lakehouse**

O Data Lakehouse (DLH) [24] [1] surge como uma abordagem híbrida, prometendo trazer as funcionalidades de gerenciamento e organização do DW para o ambiente mais flexível e de baixo custo do DL. Isso inclui suporte a transações ACID, controle de metadados e evolução de esquema.

Os DLHs são separados em 3 camadas: armazenamento, metadados e computação. O armazenamento provê a persistência de grandes volumes de informações de maneira escalável e de fácil acesso, usualmente implementado com *Object Stores*. A camada de metadados provê governança, controle de qualidade e garantias ACID. Já a camada de computação é responsável pelo processamento das cargas de trabalho.

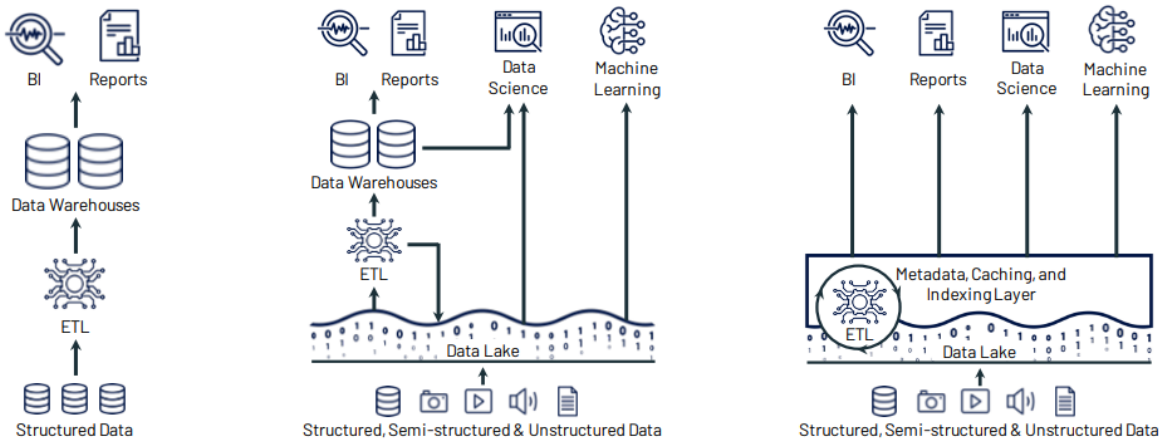


Figura 1: Estrutura de DW e DL em comparação com DLH. Fonte: Armbrust et al. (2021)

As principais ferramentas desse contexto estão na camada de metadados e são oferecidas como formatos abertos de tabela (Open Table Format do inglês). Entre as soluções *Open Source* mais populares, estão Apache Hudi [25], Apache Iceberg [26] e Delta Lake [27]. Cada um desses formatos busca aprimorar as tabelas Hive com funcionalidades específicas para cada caso de uso.

- **Hudi**: foi criado pela Uber para suportar um alto volume de *streaming*.
- **Iceberg**: desenvolvido pela Netflix, que precisava otimizar suas consultas em bases com petabytes de dados.
- **Delta Lake**: surgiu na Databricks, onde se buscava a integridade das informações por meio de operações ACID.

## 2.8 **Iceberg**

O Apache Iceberg se destaca entre os demais formatos de tabela , com adoção por grandes líderes da tecnologia como Adobe, Netflix e Apple [28] [29] [30]. Até mesmo ”rivais”como Databricks já estão dando suporte à ferramenta [31]. Entre suas principais vantagens estão: comunidade de colaboradores diversa; implementação de código aberto; integração fácil e simultânea com sistemas heterogêneos; flexibilidade na manipulação e análise dos dados. A ferramenta foca na performance analítica de altos volumes, suportando transações, upserts e deletes e evoluções de schema. [2]

Sua implementação inicial visava corrigir alguns problemas na definição das tabelas Hive. A ideia era conseguir registrar todos os arquivos contidos na tabela, de forma a permitir a implementação de transações, evitar o uso excessivo de operações de listagem no sistema de arquivos e conseguir versionar a tabela como um todo.

Além das mudanças estruturais, foram implementadas diversas pequenas mudanças que facilitam a escalabilidade e a usabilidade das tabelas, entre elas:

- **Hidden Partitioning**: poder utilizar funções de transformação sobre colunas para particionar a tabela. No Hive é necessário existir uma coluna física com a transformação aplicada, resultando na duplicação de dados e exigindo expertise na escrita de consultas.
- **Time Travel**: conseguir consultar o estado da tabela em momentos passados.
- **Rollback**: reverter a tabela a um estado anterior.
- **Mudanças de esquema**: o Iceberg consegue mudar o esquema da tabela de forma segura e consistente sem que os arquivos de dados precisem ser reescritos.

### 2.8.1 Estrutura das tabelas

De acordo com a especificação mais recente do Apache Iceberg [32], as tabelas são definidas em 2 camadas de arquivos: a camada de dados e a camada de metadados. Essa arquitetura pode ser observada na Figura 2.

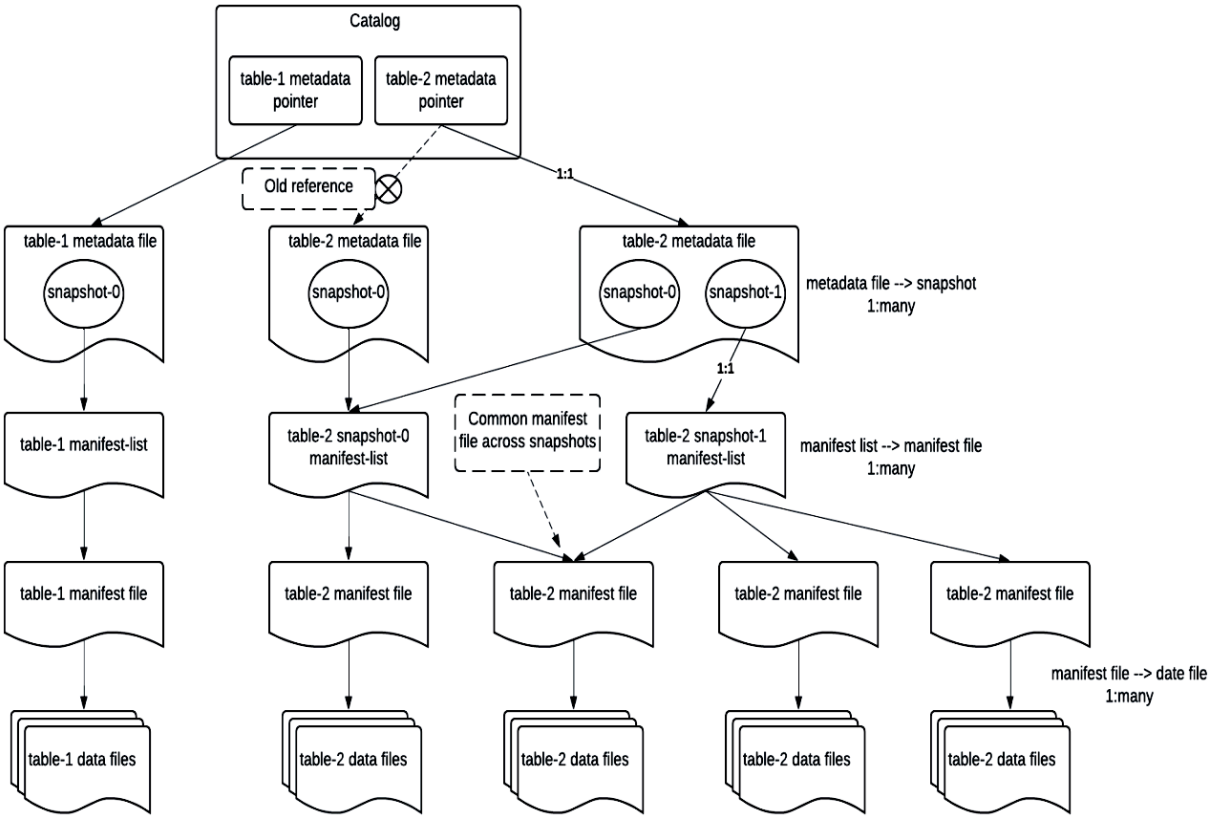


Figura 2: Especificação do formato Iceberg. Fonte: Saha (2024)

#### 2.8.1.1 Camada de metadados

Para cumprir seu objetivo de gerenciar todo o estado da tabela, o Iceberg implementa uma árvore de arquivos que guarda informações de metadados da tabela. Essa camada é composta por: arquivos de metadados, arquivos de listas de manifesto e arquivos de manifesto.

Cada estado, ou versão, da tabela é representado por uma *snapshot*. Cada *snapshot* aponta para uma lista de manifestos, que guardam todos os arquivos de manifesto associados com aquela *snapshot*, assim como diversas informações sobre partições e range de valores dos dados. Já os arquivos de manifesto mantêm uma lista de arquivos que guardam os dados, assim como diversas estatísticas sobre as colunas.

Várias dessas estatísticas são cruciais para um melhor planejamento e execução das *queries* por meio de *file pruning*.

Os arquivos de metadados, por sua vez, guardam uma lista de todas as *snapshots* da tabela. Ao ser feito um *commit*, um novo arquivo de metadado é criado, contendo uma nova *snapshot*, e o catálogo aponta para esse novo arquivo. Essa implementação garante consistência na leitura mesmo enquanto um *commit* está sendo feito.

#### 2.8.1.2 Camada de dados

Essa camada possui os dados em si da tabela, podendo ser escrita em diversos formatos como ORC, Parquet e Avro. Cada um desses arquivos pode compor uma ou mais snapshots, o que torna possível o versionamento da tabela com mínima redundância.

#### 2.8.1.3 Gerenciamento e otimização

Cada modificação feita na tabela Iceberg gera diversos novos arquivos de dados e metadados, deixando os antigos ainda disponíveis para *rollbacks* e operações *time-travel*. Para limpar arquivos antigos ou fora de uso, pode ser usado o procedimento *VACUUM* que irá expirar *snapshots* antigas e remover arquivos que não estão vinculados a nenhuma *snapshot*.

Mudanças nos dados também podem gerar uma quantidade maior de arquivos do que o desejado. Isso é comum em operações de *insert*, *delete* ou *update*. Ao invés de reescrever todos os dados em arquivos maiores e otimizados, o Iceberg escreve apenas o que foi alterado em arquivos menores chamados delta. Essa fragmentação do dataset em muitos arquivos pode degenerar a performance da tabela, já que mais arquivos deverão ser acessados para realizar uma consulta. Esse cenário pode ser remediado através de operações de *OPTIMIZE*, onde os arquivos são compactados para otimizar seus tamanhos. Essa funcionalidade, porém, pode ser bem custosa, já que exige a reescrita de diversos arquivos possivelmente grandes.

Listing 2.1: Operações de VACUUM e OPTIMIZE no Iceberg

```
VACUUM table_name;
OPTIMIZE table_name REWRITE DATA USING BIN_PACK [WHERE ...];
```

## 2.9 **Query Engines**

As *Query Engines* são ferramentas que auxiliam no acesso e manipulação de dados distribuídos, especialmente por meio da linguagem SQL. Elas agem como uma camada de virtualização, permitindo consultas a datasets em diferentes locais sem a necessidade de um ETL (*Extract, Transform, Load*), abstraindo as complexidades de consultas e agregações em diversos nós do cluster e garantindo fácil escalabilidade.

Essas soluções trabalham na camada de computação do DLH, sendo abertas para integração com uma ou várias soluções de armazenamento e catalogação, como Metastores e SGBDs. Entre os principais nomes desse cenário estão o Apache Spark, Presto/Trino e o próprio Hive [33]. Essas ferramentas possuem capacidade para várias cargas de trabalho, como consultas, *streaming* e até mesmo *machine learning*.

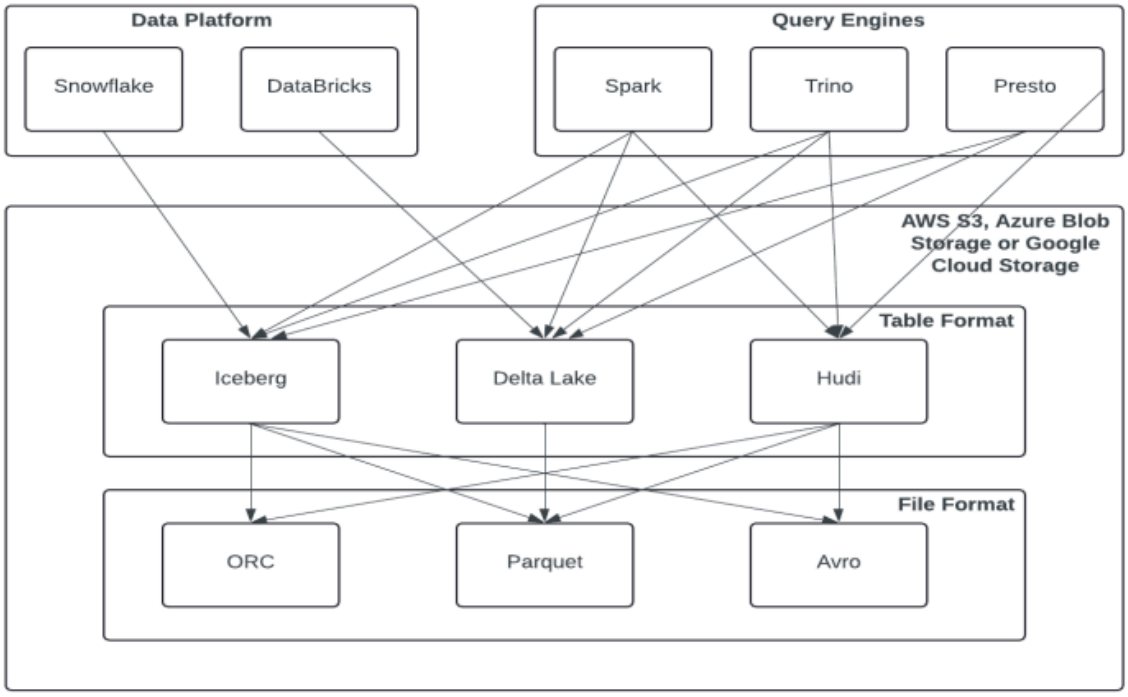


Figura 3: Composição do DLH com uso de formatos abertos de tabela e *Query Engines*. Fonte: Saha (2024)

### 2.9.1 Presto

O Presto começou no Facebook como uma alternativa ao Hive para processamento massivo de dados com alto desempenho. Foi nomeado inicialmente de PrestoDB [34], mas, após divergências com a liderança da empresa, o projeto se tornou

independente e foi renomeado para PrestoSQL e, posteriormente, Trino.

O uso de conectores permite que a ferramenta consulte dados de forma nativa em diversos sistemas como: Hive Metastore, catálogos Iceberg, SGBDs e até mesmo bancos NoSQL. Além disso, seu engenho inclui vários recursos que garantem a alta performance, como otimização de partições, execução em memória e eliminação de leitura desnecessária (*query pruning*) [35].

O Presto possui uma arquitetura distribuída composta por dois componentes principais: o nó coordenador e os nós *workers*. Enquanto o nó coordenador é responsável por gerenciar todo o fluxo de execução das consultas, como verificação de sintaxe, plano lógico e otimizações, os *workers* fazem o processamento efetivo com a leitura dos dados, agregações e junções. Essa abordagem distribuída permite que o Presto processe grandes volumes de dados de maneira eficiente e paralela.

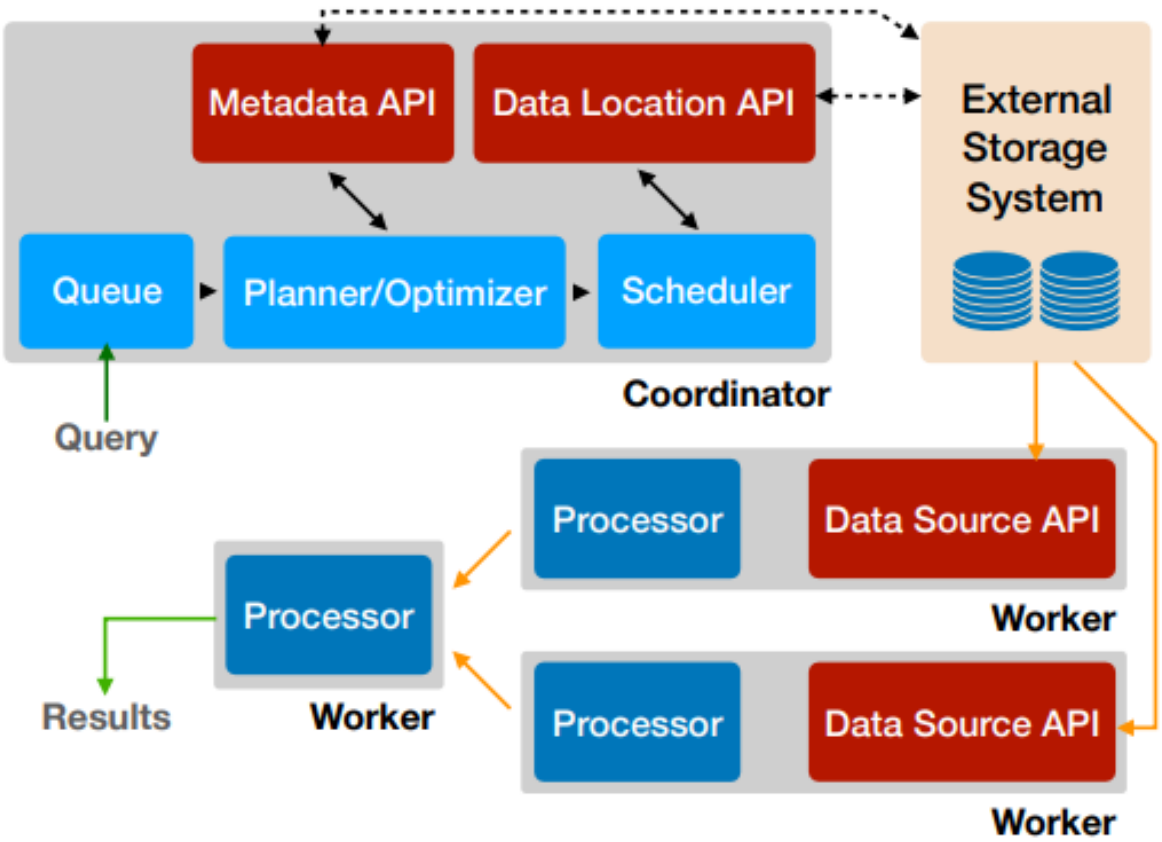


Figura 4: Arquitetura do Trino/Presto. Fonte: Sethi et al. (2019)

### 2.9.2 Amazon Athena

O Amazon Athena [36] [37] é um serviço *serverless* e interativo provido pela AWS para consultar dados armazenados no Amazon S3 utilizando SQL. Ele permite a consulta em grandes conjuntos de dados sem a necessidade de criar ou manter a infraestrutura complexa de recursos de computação dedicados.

permite a grande consulta de dados sem necessidade de acoplamento a recurso computacional especificado

Por trás do Athena está uma versão customizada do Presto, que é disponibilizada de forma *serverless* para os usuários. A cobrança desse serviço é feita em cima da quantidade de dados acessada em cada consulta. Usuários ficam, então, incentivados a gerar consultas performáticas e otimizar o uso da compressão de arquivos.

O serviço é automaticamente integrado com o AWS Glue Data Catalog [38], que age como um *metastore* para a solução. Isso facilita a integração de dados armazenados no S3 com o ambiente de consultas, que pode ser configurado em apenas alguns minutos. Assim como o Presto, o Athena também provê integrações com outros serviços de armazenamento por meio de conectores como JDBC.

O Athena é largamente utilizado para *queries ad hoc*, possibilitando a analistas e engenheiros explorar e transformar os dados armazenados sem o uso de um ETL nem gerenciamento de infraestrutura. Ao serem disparadas, as consultas entram primeiramente em uma fila onde esperam recursos estarem disponíveis para seu processamento iniciar. Esse fato é importante para qualquer mensuração feita, já que o tempo aguardando na fila não está atrelado à performance da consulta em si.

## 2.10 **SSB**

Existem vários benchmarks propostos para ambientes de DW, sendo vários dos mais aceitos formulados pelo Transaction Processing Performance Council (TPC), como exemplo do TPC-H e do TPC-DS. Se destaca também o Star Schema Benchmark (SSB), que foca no uso do esquema estrela.

O TPC-H [39] é um benchmark que simula o processo de suporte à decisão em cima de dados históricos de pedidos e vendas de uma companhia em um certo período. Ele foca em *queries* complexas que avaliam o sistema em uma variedade de

cargas de trabalho. Seu esquema tem uma arquitetura floco de neve com 1 tabela fato e 7 dimensões, além de 22 consultas que analisam diferentes aspectos da ferramenta usando a métrica de consultas por hora (Qph@SF), onde SF é o fator de escala do *dataset* gerado.

Já o TPC-DS [40] se apresenta como uma alternativa mais complexa e completa ao TPC-H. Sendo focado numa gama maior de operações, ele apresenta uma arquitetura de 7 tabelas fato e 17 dimensões com uma mistura de esquema estrela e floco de neve. Por fim, são propostas 99 *queries* para avaliar a performance da solução.

Também derivado do TPC-H por pesquisadores da Universidade de Massachussets, o Star Schema Benchmark (SSB) [4] simula um cenário mais específico. Ao invés de um cenário de *queries ad hoc* em um esquema floco de neve, o SSB foca em uma avaliação direta dos modelos dimensionais ao trabalhar com o esquema estrela. Nessa arquitetura, existe 1 tabela fato e 4 dimensões. O esquema do SSB pode ser visto na Figura 5.

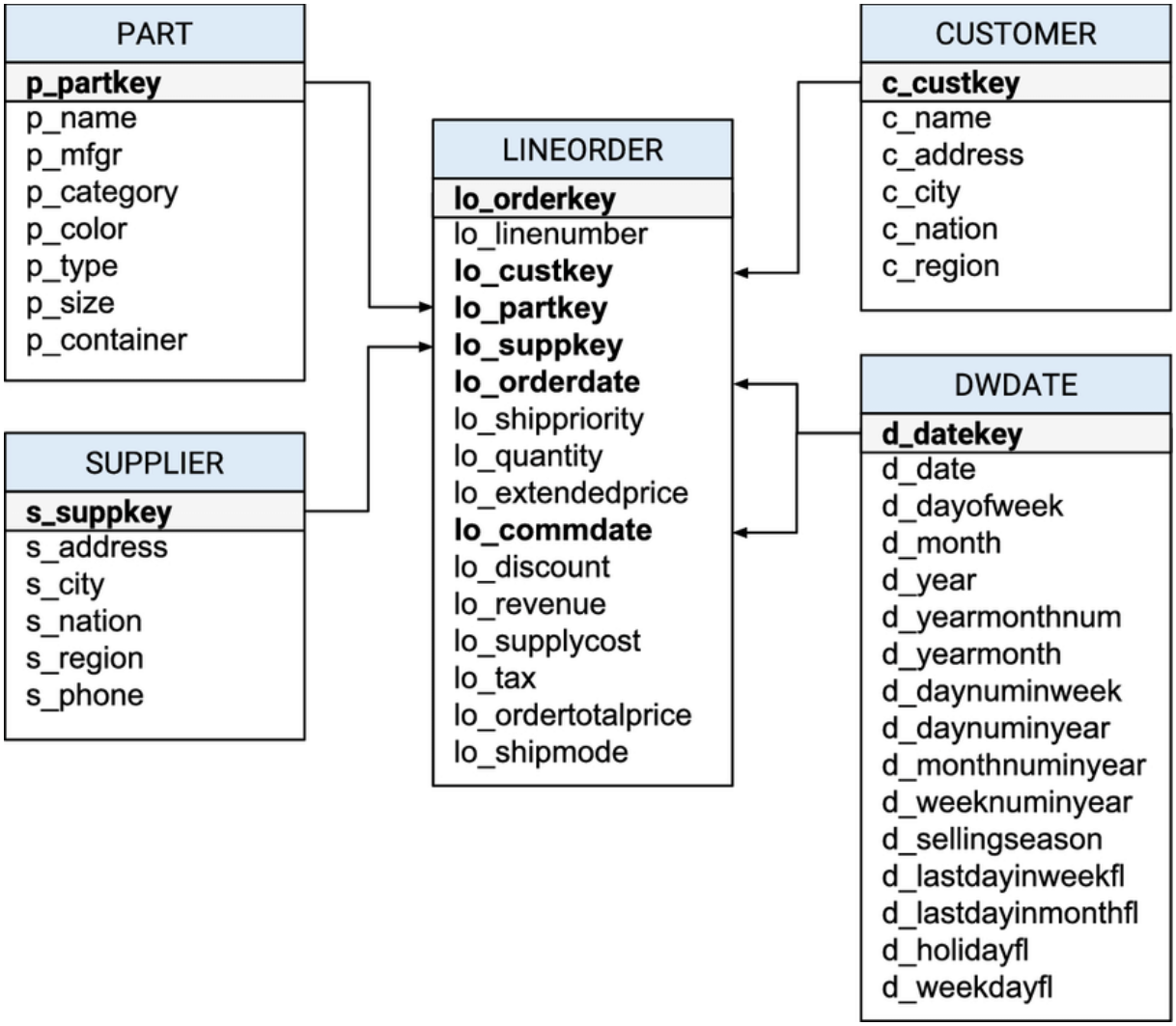


Figura 5: Modelo do SSB.

Neste trabalho, o SSB foi escolhido por apresentar uma solução focada no esquema estrela, um esquema bem estabelecido na modelagem de DW [41]. Por ter uma arquitetura simplificada, ele permite uma análise mais transparente e compreensível, além de se aproximar mais fielmente do contexto de diversas aplicações reais.

Os autores do SSB propuseram 13 consultas que representam operações comuns em um DW como *roll up*, *slice* e *dice*. Essas consultas são separadas em grupos, onde cada grupo possui operações de diferentes complexidades. Dentro de cada grupo, as *queries* são distinguidas pelos filtros aplicados, resultando no fator de seletividade para cada uma delas. A seletividade e o número de *joins* em cada *query* são demonstrados na tabela Tabela 1.

Tabela 1: Seletividade das Queries do SSB.

| Query | # de Joins | Cálculo de seletividade | Seletividade (%) | Classe seletividade |
|---|---|---|---|---|
| **Q1.1** | 1 | $\frac{1}{7} \times \frac{3}{11} \times 0.47$ | $1.9 \times 10^{-2}$ | Baixa |
| **Q1.2** | 1 | $\frac{1}{84} \times \frac{3}{11} \times \frac{2}{10}$ | $6.5 \times 10^{-4}$ | Média |
| **Q1.3** | 1 | $\frac{1}{10} \times \frac{3}{11} \times \frac{1}{364}$ | $7.5 \times 10^{-5}$ | Alta |
| **Q2.1** | 3 | $\frac{1}{125} \times \frac{1}{5}$ | $8.0 \times 10^{-3}$ | Baixa |
| **Q2.2** | 3 | $\frac{1}{125} \times \frac{1}{5}$ | $1.6 \times 10^{-3}$ | Média |
| **Q2.3** | 3 | $\frac{1}{1000} \times \frac{1}{5}$ | $2.0 \times 10^{-4}$ | Alta |
| **Q3.1** | 3 | $\frac{1}{5} \times \frac{1}{5} \times \frac{6}{7}$ | $3.4 \times 10^{-2}$ | Baixa |
| **Q3.2** | 3 | $\frac{1}{25} \times \frac{1}{25} \times \frac{6}{7}$ | $1.4 \times 10^{-3}$ | Média |
| **Q3.3** | 3 | $\frac{1}{125} \times \frac{1}{125} \times \frac{6}{7}$ | $5.5 \times 10^{-5}$ | Média |
| **Q3.4** | 3 | $\frac{1}{84} \times \frac{1}{125} \times \frac{1}{125}$ | $7.6 \times 10^{-7}$ | Alta |
| **Q4.1** | 4 | $\frac{2}{5} \times \frac{1}{5} \times \frac{1}{5}$ | $1.6 \times 10^{-2}$ | Baixa |
| **Q4.2** | 4 | $\frac{2}{7} \times \frac{2}{5} \times \frac{1}{5} \times \frac{1}{5}$ | $4.6 \times 10^{-3}$ | Média |
| **Q4.3** | 4 | $\frac{2}{7} \times \frac{1}{25} \times \frac{1}{25} \times \frac{1}{5}$ | $9.1 \times 10^{-5}$ | Alta |

O SSB pode ser customizado a partir do parâmetro *Scale Factor* (SF), que dita o tamanho da massa de dados gerada. A quantidade de linhas de cada tabela apresenta um comportamento diferente em relação ao aumento do SF. A tabela fato *lineorder* é, naturalmente, a que mais cresce com o fator. Já a tabela de datas não sofre alterações, sempre apresentando dados dos mesmos 7 anos. A relação de linhas para cada tabela dependendo do SF pode ser vista na Tabela 2.

Tabela 2: Linhas por tabela por Scale Factor

| Tabela | Número de linhas |
|---|---|
| **Part** | $200000 \times (1 + log_2(SF))$ |
| **Supplier** | $SF \times 2000$ |
| **Date** | 7 anos em dias |
| **Customer** | $SF \times 30000$ |
| **Lineorder** | $SF \times 600000$ |

# 3 TRABALHOS RELACIONADOS

Este capítulo explora trabalhos relacionados com a tese apresentada. Cada seção irá abordar um trabalho específico e suas contribuições para a tese atual.

## 3.1 O trabalho de Vargas (2022)

Vargas (2022) [42] aborda a comparação de performance entre 3 formatos de tabelas para Data Lakehouse: Apache Hudi, Apache Iceberg e Delta Lake. O trabalho apresenta 2 experimentos distintos: um inspirado no TPC-DS que realiza as operações descritas no *benchmark*; e outro que analisa o impacto que *updates* na tabela fato têm no resultado das consultas. O tamanho do *cluster* é também uma variável considerada nos experimentos, sendo realizado com diferentes números de máquinas e capacidades de memória.

As principais métricas consideradas são o tempo de execução médio das consultas, assim como uma métrica chamada *speedup*, que analisa o aumento de performance com maior número de nós no *cluster*. O tempo de carregamento das tabelas também é metrificado.

O desenho do segundo experimento, com o uso de *updates* na tabela fato, foi usado como base para o desenvolvimento da metodologia de um dos experimentos desta tese.

## 3.2 O trabalho de Hartzell (2023)

Hartzell (2023) [33] compara diversas soluções de processamento para ambientes de Big Data. A análise visa identificar vantagens e desvantagens de cada solução, além de sumarizar casos de uso para cada uma. Dentre as soluções exploradas estão: AWS Athena; AWS EMR com Spark; AWS EMR com Hive; e AWS Glue.

Ao final do trabalho é apresentada uma tabela com um comparativo geral entre as soluções, indicando: casos de uso, modelos de precificação, modo de uso e um breve comentário sobre desempenho. O trabalho não traz um comparativo detalhado de performance e benchmarking, focando mais nas linhas gerais das implementações.

A exposição das características de cada ferramenta foi utilizada como base de

escolha da *engine* para realização dos testes entre Hive e Iceberg. O passo a passo da configuração da ferramenta selecionada também foi inspirado no que foi apresentado no artigo.

## 3.3 **O trabalho de Rabelo Ferreira et al. (2024)**

Rabelo Ferreira et al. (2024) [43] aborda o impacto da desnormalização de dados em sistemas OLAP e HTAP, visando um uso eficiente de arquiteturas *Data Warehouse as a Service*. Nele, é criada uma rotina de testes usando o *dataset* do SSB em sua forma padrão e em uma forma customizada de *flat table*, assim como variação do *Scale Factor* entre SF10 e SF100.

Na arquitetura das soluções também foi levado em consideração o *dataset* caber ou não totalmente na memória do *cluster*, gerando cenários em cada um dos casos. Embora isso seja importante em cenários de DWs tradicionais, *Query Engines* modernas no Hadoop já utilizam técnicas de *file* e *partition pruning* [35], o que diminui massivamente o impacto que a memória tem no resultado.

O design do experimento como um todo visa, também, a simplicidade na implementação, buscando um cenário mais acessível e menos excessivamente otimizado. Essa consideração de valor na perspectiva do praticante da engenharia de dados serviu como base para as escolhas tecnológicas para os experimentos deste trabalho.

## 3.4 **O trabalho de Nelluri et al. (2025)**

Nelluri et al. (2025) [44] explora as diferenças entre diferentes formatos de tabela para Big Data. O estudo visa explorar características, vantagens e limitações dos formatos ORC, Parquet, AVRO e Iceberg, gerando base para melhores decisões tecnológicas. Foram avaliadas várias métricas, como eficiência de armazenamento, performance em consultas, evolução de esquema e compatibilidade com ferramentas analíticas.

As consultas foram feitas utilizando as engines Spark-SQL e Hive, e separadas em consultas de leitura e escrita. Foi encontrado que o Iceberg possui vantagem na escrita comparado com o Parquet; contudo, a leitura no total foi aproximadamente 30% pior. Foi notado, porém, que o Iceberg leva vantagem em cenários com particiona-

mento. A eficiência de armazenamento das 2 soluções foi elencada como semelhante.

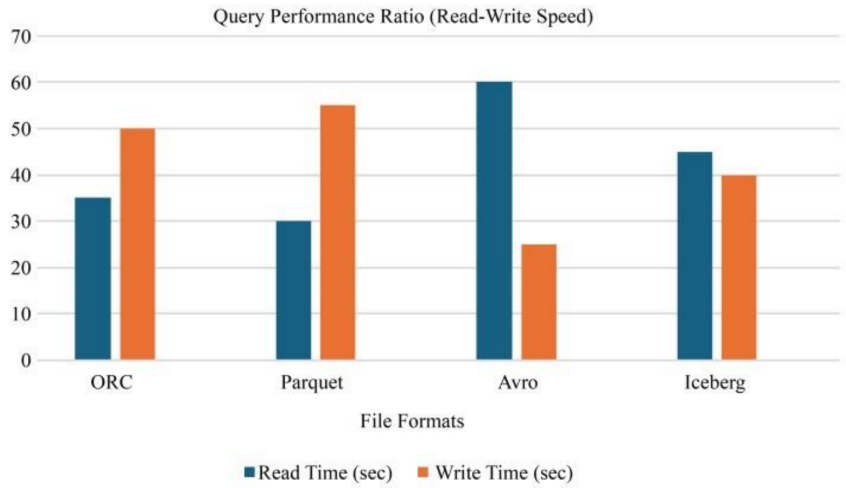


Figura 6: Performance comparada de leitura e escrita. Fonte: Nelluri et al. 2025

Esses resultados serviram como base para avaliar a performance comparativa entre tabelas Iceberg e Hive, comparáveis às tabelas Parquet do estudo mencionado.

# 4 METODOLOGIA

Neste capítulo é detalhada a metodologia usada para conduzir os experimentos de comparação entre as tabelas Hive e Iceberg. São apresentados os objetivos dos testes, procedimentos de coleta e tratamento dos dados, definição e configuração do ambiente de testes, escolha de cargas de trabalho, métodos de avaliação, assim como limitações.

## 4.1 Objetivo

O objetivo principal do experimento é comparar a performance analítica de tabelas Iceberg e Hive em uma arquitetura de esquema estrela. Em sequência, é analisado o impacto que alterar os dados da tabela, gerando uma fragmentação do *dataset*, tem no resultado das consultas.

Para todo o design do experimento, foi pensado em um cenário onde os engenheiros responsáveis buscam soluções de fácil implementação, fácil manutenção, baixo custo e de acesso democrático dentro da organização.

## 4.2 Hipóteses

- **H0 (Hipótese Nula)**: Não existe diferença significativa de performance entre tabelas Hive e tabelas Iceberg nos cenários avaliados.
- **H1 (Hipótese Alternativa)**: Existe diferença significativa entre a performance entre tabelas Hive e tabelas Iceberg nos cenários avaliados.

## 4.3 Variáveis experimentais

Variáveis dependentes:

- **Tempo médio de processamento**: tempo médio que cada consulta durou descontado do tempo de fila para todas as 30 execuções de cada *query*, assim como o intervalo de confiança de 95%.
- **Quantidade de bytes lidos**: quantidade de bytes lido em cada consulta.

Variáveis independentes:

- **Scale Factor**: SF10 e SF100
- **Tipo de tabela**: Hive e Iceberg
- **Tipo de carga**: carga completa, cargas incrementais e cargas incrementais com OPTIMIZE
- **Consulta SSB**: cada uma das 13 consultas definidas no *benchmark*.

## 4.4 Geração do dataset

As tabelas base do Star Schema Benchmark foram geradas utilizando a ferramenta DBGen em uma máquina virtual utilizando os fatores SF10 e SF100. Os dados foram então formatados utilizando DuckDB [45] e feito o upload para o Amazon S3 em formato parquet. Esses arquivos, então, ficaram disponíveis para carregar as tabelas usadas no experimento. Em formato parquet, os datasets SF10 e SF100 ocuparam 2.1 GB e 22 GB respectivamente. O passo a passo para essa operação foi descrito em [46].

Embora as ferramentas testadas sejam feitas para trabalharem com TBs de dados, foram utilizados os fatores SF10 e SF100. Essa escolha, em conformidade com Ferreira Rabelo et al. 2024 [43], visa testar volumes de dados que consigam estressar as soluções, mas que não gerem custos demasiados para a realização dos experimentos. Além disso, os *datasets* escolhidos refletem volumetrias que podem ser encontradas no dia-a-dia, fornecendo *insights* para cenários com quantidades menores de dados, mas que ainda necessitam de aplicações de Big Data para serem processados.

## 4.5 Arquitetura utilizada nos testes

A arquitetura utilizada visa replicar um cenário que utiliza uma *query engine* distribuída e operando em ambiente em nuvem. Foram escolhidos serviços de fácil implementação e autogerenciados, representando casos reais onde os engenheiros buscam soluções de baixo custo e manutenção.

Para a camada de computação foi escolhido o Amazon Athena [11], onde é possível utilizar SQL para manipular e explorar os dados de um Data Lake. Sendo um

fork do Presto, ele provê grande poder computacional, mas de forma *serverless*, onde não é preciso gerenciar a infraestrutura associada ao *cluster*. Essa escolha vem baseada em tendências na direção de soluções *serverless* para análises de Big Data [47].

Como catálogo de dados, foi escolhido o AWS Glue Data Catalog [38]. O Glue Data Catalog provê um serviço de metadados similar ao Hive Metastore, escalável e compatível com fontes de dados como AWS S3, bancos relacionais e bancos NoSQL. Nele é possível definir tabelas e esquemas que podem ser consultados de forma nativa por ferramentas como AWS Athena e AWS Redshift, além de utilizar um adaptador para Hive, que o torna acessível para *query engines* como Apache Spark e Trino. Este catálogo possui suporte tanto para tabelas Hive quanto para Iceberg.

A camada de armazenamento foi, então, naturalmente designada para o AWS S3, a oferta de *Object Storage* da AWS. Para garantir o máximo de performance, todos os serviços foram configurados para a mesma região no us-east-2.

## 4.6 **Processo experimental**

Com os dados crus carregados no S3, foram criadas tabelas Hive externas, que servem como base para carregar as tabelas em que são feitos *benchmarks*. O carregamento é feito através de operações *INSERT INTO ... SELECT*.

Listing 4.1: Operação de carregamento das tabelas do experimento

```
INSERT INTO tabela_experimento SELECT * FROM tabela_externa;
```

São então selecionados o scale factor (SF10 ou SF100) e o tipo de tabela (Iceberg ou Hive) a serem testados. As tabelas de teste são criadas e carregadas a partir das tabelas externas e executadas cada uma das consultas previstas no SSB. As consultas são rodadas 30 vezes para garantir resultados estatisticamente confiáveis segundo o teorema do limite central, mas mitigando os custos do experimento. Em todos os testes, foi utilizado o formato padrão em parquet [48] [9] para armazenamento dos dados.

É importante ressaltar que o SSB requer a remoção dos dados salvos em cache para não enviesar os resultados. O Athena faz apenas *caching* da mesma *query* por um período curto de tempo, o que foi desabilitado. *Caching* de arquivos ou de *query plans* não são suportados pela plataforma. Já o S3 e Glue Data Catalog não implementam

camadas de cache. Dessa forma, as *queries* podem ser executadas em sequência sem riscos para a qualidade dos resultados.

Para avaliar o impacto que inserções têm na performance analítica das tabelas, foi feita uma modificação no processo de carga da tabela gerando um segundo experimento. Nele, os dados da tabela fato são carregados em 100 cargas incrementais, cada uma contendo 1% da base. Após a carga, as *queries* são executadas 30 vezes e, ao final, as tabelas Iceberg são otimizadas e os testes rodados novamente. No processo de otimização, os arquivos são compactados e reescritos, que deve, em teoria, restaurar o impacto de performance feito pela fragmentação dos dados. As tabelas Hive não foram otimizadas, já que a reescrita dos arquivos resultaria no mesmo resultado do primeiro experimento.

O procedimento experimental completo pode ser visualizado na Figura 7.

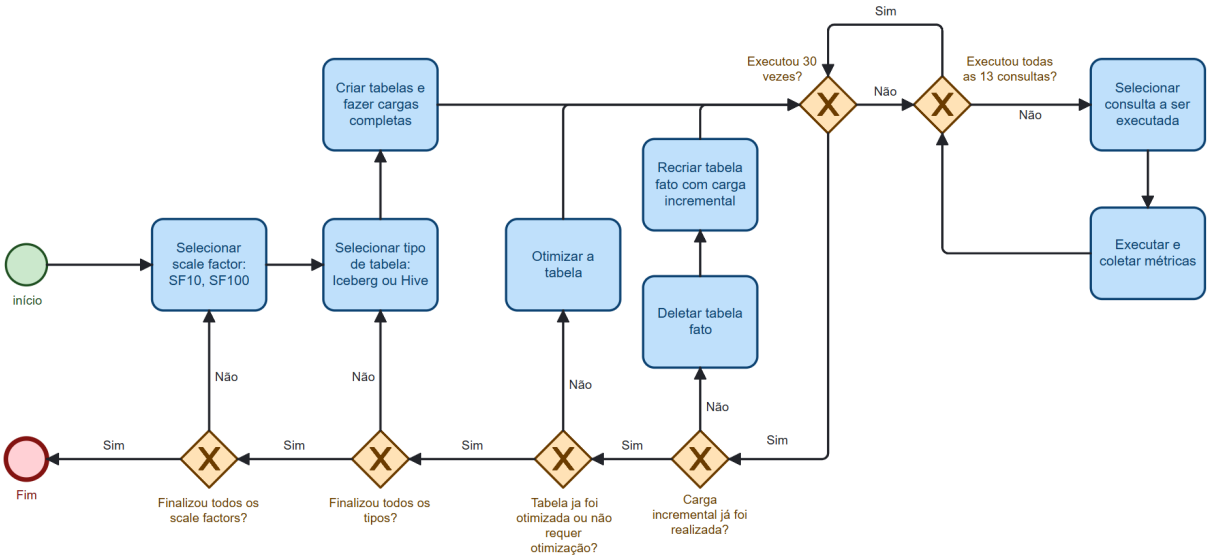


Figura 7: Diagrama do experimento

Foi escolhido o procedimento de *insert* ao invés de *update* pois a segunda operação, embora funcione para o Apache Iceberg, não possui suporte para Hive dentro do ambiente do Athena. Essa escolha, porém, continua gerando um cenário de múltiplos arquivos que é o foco do experimento.

A carga incremental só foi feita na tabela fato, já que é a que mais contribui para a volumetria e tende a ter maior número de alterações durante o tempo. As tabelas dimensão, além de sofrerem menos alterações, possuem volumetria menor, o que torna processos reescrita menos custosos.

## 4.7 Coleção das métricas

Foi configurado um *script* python que utiliza o Athena para rodar os experimentos e salvar as métricas [46]. Para cada consulta, foram mensuradas 4 métricas: tempo de processamento da consulta, tempo de planejamento, tempo de execução e quantidade de bytes lidos. O tempo de processamento foi calculado a partir do tempo total da consulta, descontado do tempo em que a requisição ficou na fila aguardando o processamento. Todas as métricas cruas foram extraídas diretamente do cursor retornado ao executar a *query*.

O tempo de processamento foi tomado como a principal medida de performance das soluções. A quantidade de bytes lidos foi considerada um medidor de performance secundário, já que influencia diretamente nos custos de operação do Athena. Já os tempos de planejamento e execução serviram para melhor elucidar as possíveis diferenças entre as soluções.

O tamanho total da tabela não foi avaliado por ser mais dependente do formato de arquivo da camada de dados do que do tipo de tabela. De acordo com [44], os tamanhos de tabelas Iceberg e Hive foram semelhantes. Além disso, as tabelas Iceberg armazenam todo o histórico da tabela, não só o estado atual, fazendo a comparação desigual.

## 4.8 Análise dos resultados

Utilizando um script python [46], as métricas de tempo de processamento e bytes lidos foram tratadas através de suas médias e intervalos de confiança de 95%. As demais métricas foram tratadas através de suas médias e usadas para enriquecer o entendimento dos resultados obtidos, juntamente com o recurso *EXPLAIN ANALYZE* provido pelo Athena.

Para avaliar a performance de forma unificada, foi utilizada a Performance Relativa (PR). Nela, a métrica do Iceberg é dividida pela sua contraparte no Hive, demonstrando o percentual do ganho em cada uma das *queries*. Com esse indicador, é possível comparar diretamente os resultados de consultas diferentes e obter um resultado final intuitivo. Usualmente, é usado o termo *Query Speedup* (QS) para essa estatística, mas, como esse tratamento será também aplicado à métrica de bytes lidos,

o termo não seria apropriado.

$$PR = \frac{Iceberg}{Hive} \tag{4.1}$$

Aplicando um teste t de uma amostra, é possível obter comprovação estatística da diferença de performance entre as 2 soluções. O valor de referência usado foi de 1, caso onde a performance das ferramentas é igual. Qualquer valor acima de 1 atesta vantagem para o Hive, já valores menores que 1, para o Iceberg.

O teste t foi aplicado utilizando a biblioteca pingouin e analisados os indicadores de valor-p e intervalo de confiança, enquanto o valor médio da Performance Relativa já indica o percentual de ganho ou perda de performance. O teste ANOVA não foi necessário, pois em cada um dos múltiplos experimentos havia apenas 2 soluções a serem testadas.

## 4.9 **Potenciais limitações**

- A escolha de query *query engine* pode impactar diretamente nos resultados, outras opções como Spark e Trino poderiam ser usadas e gerar resultados diferentes.
- O Apache Iceberg foi construído pensando em melhorar a escalabilidade do Hive ao chegar em Petabytes de dados com múltiplas partições, contudo, os presentes experimentos foram realizados em *datasets* menores de 100GB. O aumento de volumetria pode revelar alterações nos resultados.
- Diferentes catálogos de dados podem ter diferentes funcionalidades e otimizações que alteram a performance das consultas. Outras opções como Hive Metastore, Unity Catalog ou Nessie poderiam servir como alternativas.
- Tanto o Glue Data Catalog quanto o Athena possuem implementações proprietárias e não suportam certas operações nas tabelas Hive e Iceberg. Utilizar uma combinação *open source* de *metastores* e *query engine* podem trazer outras opções para manipulação dos dados e definições das tabelas.
- Foi escolhido o número de 100 cargas incrementais para o segundo experimento. É possível que esse número seja insuficiente para gerar mudanças expressivas no resultado das *queries*.

- Nesse trabalho não foram utilizadas técnicas de particionamento na definição das tabelas. Essas técnicas são bastante comuns em ambientes transacionais e tem grande impacto na performance das consultas.

# 5 RESULTADOS

Após a execução dos testes, foi verificada uma divergência em algumas métricas retornadas pelo Athena. Para algumas consultas, o tempo de planejamento e execução passava do tempo total. Esse comportamento foi confirmado através do console gráfico do Athena. Uma possível explicação são as aproximações usadas para gerar uma métrica única a partir de vários nós. Como os tempos de planejamento e execução são apenas variáveis usadas no enriquecimento do experimento, eles foram mantidos.

## 5.1 Resultados para carga padrão

### 5.1.1 SF10

A Figura 8 apresenta os resultados médios com carga padrão comparando o Iceberg com o Hive no SF10. Houve grande diferença no tempo de processamento em todas as consultas, com uma clara vantagem para o Hive. Nota-se que o Iceberg também apresentou resultados menos constantes, refletidos no maior intervalo de confiança. A maior variação das *queries* do grupo 4 no geral foi atribuída à natureza serverless do Athena.

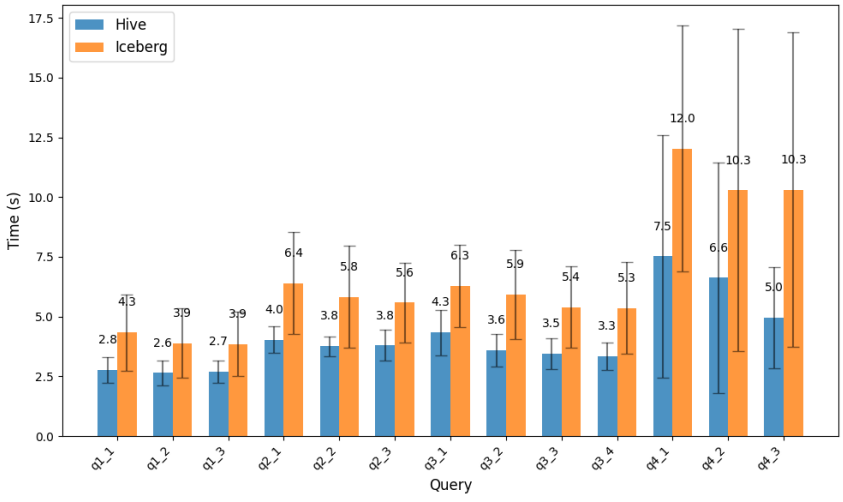


Figura 8: Tempo de processamento para Iceberg e Hive com carga padrão e SF10.

A quantidade de bytes lidos é similar, com alguns casos sendo menores para o Iceberg. O tempo de planejamento das consultas foi semelhante, com menos de 15% de diferença. No total, o Iceberg foi 57% mais lento, com uma diferença menor de 1%

na quantidade de dados lidos.

Tabela 3: Média das métricas de execução para SF10 com carga padrão

| Query | Hive | | | Iceberg | | |
|---|---|---|---|---|---|---|
| | T. Proces.(s) | Bytes Lidos(MB) | T. Planej.(s) | T. Proces.(s) | Bytes Lidos(MB) | T. Planej.(s) |
| **Q1.1** | 2.85 | 437.19 | 0.38 | 4.05 | 446.67 | 0.33 |
| **Q1.2** | 2.78 | 437.19 | 0.39 | 4.31 | 446.67 | 0.36 |
| **Q1.3** | 2.85 | 437.19 | 0.38 | 4.31 | 446.67 | 0.38 |
| **Q2.1** | 4.13 | 673.45 | 0.47 | 6.10 | 672.15 | 0.53 |
| **Q2.2** | 3.86 | 675.72 | 0.47 | 5.95 | 653.85 | 0.55 |
| **Q2.3** | 3.82 | 669.87 | 0.46 | 5.85 | 648.36 | 0.55 |
| **Q3.1** | 4.33 | 575.12 | 0.48 | 6.61 | 577.98 | 0.73 |
| **Q3.2** | 3.74 | 582.56 | 0.49 | 5.80 | 574.79 | 0.57 |
| **Q3.3** | 3.53 | 578.03 | 0.48 | 5.53 | 571.24 | 0.55 |
| **Q3.4** | 3.51 | 584.57 | 0.50 | 5.54 | 590.97 | 0.55 |
| **Q4.1** | 8.25 | 850.49 | 0.39 | 12.18 | 846.41 | 0.51 |
| **Q4.2** | 7.02 | 853.29 | 0.44 | 11.46 | 844.47 | 0.55 |
| **Q4.3** | 5.62 | 854.44 | 0.50 | 10.95 | 840.87 | 0.56 |
| **Total** | 56.16 | 8209.11 | 5.84 | 88.64 | 8161.10 | 6.72 |

Na Tabela 4 é possível ver o resultado dos testes agregados para todas as queries com uso da métrica de performance relativa. Os resultados comprovam a grande vantagem do Hive na velocidade, mas indicam uma paridade em termos de leitura de bytes. Os resultados vão de acordo com [44].

Tabela 4: Performance Relativa entre Hive e Iceberg com carga padrão e SF10.

| Métrica | PR | IC95% | valor p | Melhor performance |
|---|---|---|---|---|
| **Tempo de processamento** | 1.57 | [1.52, 1.62] | <0.001 | Hive |
| **Bytes lidos** | 1.00 | [1.00, 1.01] | 0.39 | Performance igual |

### 5.1.2 SF100

O mesmo cenário é encontrado na escala SF100. Nessa comparação, porém, o Hive obteve uma maior constância nos resultados, enquanto o Iceberg manteve um grande intervalo de confiança. O comportamento de maior variação no grupo 4 não se mostrou presente.

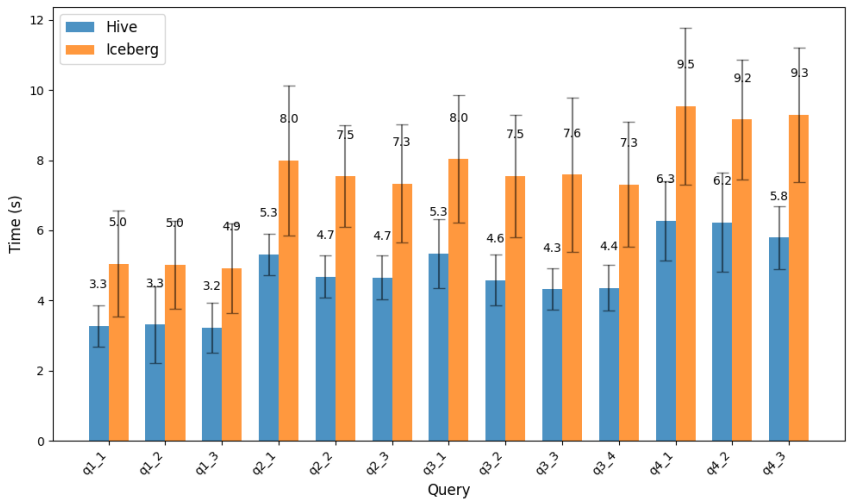


Figura 9: Tempo de processamento para Iceberg e Hive com carga padrão e SF100.

Comparado com o SF10, os bytes lidos subiram em 10x, o que é esperado para um *dataset* 10x maior. Os tempos de processamento para os grupos de *query* 1, 2 e 3 foram levemente superiores, demonstrando a boa escalabilidade no processamento.

Já para o grupo 4 houve uma redução no tempo comparado com o S10. Esse comportamento, apesar de parecer uma anomalia, esteve presente em ambas as soluções e em outros experimentos, indicando uma característica das *queries*. O plano de execução detalhado indicou que o Athena utilizou mais recursos para aquela computação, gerando um aumento de performance no tempo.

Tabela 5: Média das métricas de execução para SF100 com carga padrão

| Query | Hive | | | Iceberg | | |
|---|---|---|---|---|---|---|
| | T. Proces.(s) | Bytes Lidos(MB) | T. Planej.(s) | T. Proces.(s) | Bytes Lidos(MB) | T. Plan.(s) |
| **Q1.1** | 3.28 | 4372.87 | 0.38 | 5.05 | 4467.68 | 0.33 |
| **Q1.2** | 3.31 | 4372.87 | 0.38 | 5.02 | 4467.68 | 0.35 |
| **Q1.3** | 3.22 | 4372.87 | 0.38 | 4.92 | 4467.68 | 0.34 |
| **Q2.1** | 5.31 | 6380.02 | 0.48 | 7.99 | 6363.06 | 0.56 |
| **Q2.2** | 4.68 | 6380.04 | 0.46 | 7.54 | 6363.06 | 0.55 |
| **Q2.3** | 4.65 | 6380.04 | 0.47 | 7.33 | 6363.06 | 0.54 |
| **Q3.1** | 5.33 | 5476.31 | 0.47 | 8.03 | 5517.84 | 0.58 |
| **Q3.2** | 4.58 | 5473.45 | 0.46 | 7.55 | 5319.57 | 0.58 |
| **Q3.3** | 4.32 | 5477.46 | 0.47 | 7.59 | 5333.50 | 0.59 |
| **Q3.4** | 4.36 | 5489.05 | 0.47 | 7.31 | 5540.49 | 0.53 |
| **Q4.1** | 6.27 | 8162.56 | 0.52 | 9.54 | 8165.28 | 0.67 |
| **Q4.2** | 6.23 | 8624.48 | 0.5 | 9.16 | 8677.89 | 0.65 |
| **Q4.3** | 5.79 | 8626.35 | 0.53 | 9.29 | 8679.77 | 0.71 |
| **Total** | 61.33 | 79588.37 | 6.02 | 96.32 | 79726.56 | 6.98 |

A escolha do poder computacional é interna do Athena e não influencia no custo do processamento. Apesar de não ter sido encontrado nada na literatura sobre esse

tema, essa funcionalidade pode ser investigada para tentar melhorar a performance das consultas, mantendo o mesmo custo.

Os tempos de planejamento das queries foram similares, sendo a diferença na performance devido ao maior tempo de execução. O Hive manteve a mesma vantagem na velocidade e paridade nos dados lidos, comprovado pela Tabela 6.

Tabela 6: Performance Relativa entre Hive e Iceberg com carga padrão e SF100.

| Métrica | PR | IC95% | valor p | Melhor performance |
|---|---|---|---|---|
| **Tempo de processamento** | 1.57 | [1.52, 1.62] | <0.001 | Hive |
| **Bytes lidos** | 1.00 | [1.00, 1.01] | 0.58 | Performance igual |

## 5.2 **Resultados para carga incremental**

### 5.2.1 SF10

Na Tabela 7 são apresentados os tempos das consultas realizadas após a carga incremental da tabela fato. Aqui, o Hive amplia a diferença de performance, com o Iceberg demorando o dobro do tempo. Observando os planos de consulta, foi observado que o tempo de transferência de dados entre os estágios de processamento foi o principal responsável pela diferença. Porém, a leitura de bytes aumentou em torno de 5%.

Tabela 7: Média das métricas de execução para SF10 com carga incremental

| Query | Hive | | | Iceberg | | |
|---|---|---|---|---|---|---|
| | T. Proces.(s) | Bytes Lidos(MB) | T. Planej.(s) | T. Proces.(s) | Bytes Lidos(MB) | T. Planej.(s) |
| **Q1.1** | 1.91 | 438.22 | 0.41 | 4.02 | 448.14 | 0.37 |
| **Q1.2** | 1.84 | 438.22 | 0.39 | 4.12 | 448.14 | 0.39 |
| **Q1.3** | 1.82 | 438.22 | 0.43 | 4.07 | 448.14 | 0.39 |
| **Q2.1** | 2.69 | 685.62 | 0.50 | 5.22 | 674.17 | 0.55 |
| **Q2.2** | 2.34 | 681.70 | 0.50 | 4.96 | 669.19 | 0.57 |
| **Q2.3** | 2.28 | 681.70 | 0.48 | 4.91 | 654.23 | 0.53 |
| **Q3.1** | 3.02 | 634.45 | 0.53 | 5.33 | 635.87 | 0.55 |
| **Q3.2** | 2.55 | 642.19 | 0.52 | 5.60 | 624.53 | 0.58 |
| **Q3.3** | 2.39 | 643.67 | 0.50 | 5.10 | 626.81 | 0.64 |
| **Q3.4** | 2.56 | 642.46 | 0.67 | 4.80 | 637.22 | 0.61 |
| **Q4.1** | 3.21 | 908.53 | 0.52 | 8.91 | 894.26 | 0.50 |
| **Q4.2** | 3.95 | 904.84 | 0.53 | 7.28 | 885.94 | 0.65 |
| **Q4.3** | 3.08 | 915.30 | 0.56 | 7.02 | 883.56 | 0.63 |
| **Total** | 33.64 | 8655.12 | 6.54 | 71.34 | 8530.20 | 6.96 |

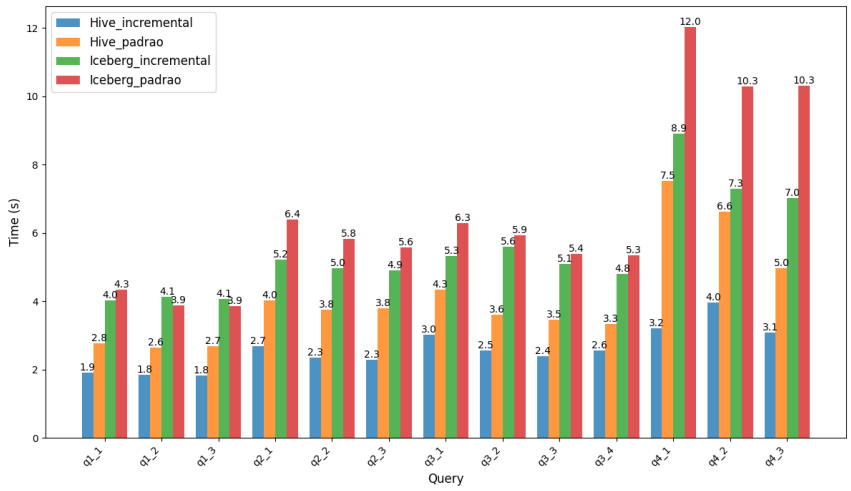


Figura 10: Comparação das cargas incremental e padrão no SF10.

Com a fragmentação do *dataset*, foi verificada uma velocidade superior no geral quando comparada com a carga simples, de 41% para o Hive e 19% para o Iceberg. A maior quantidade de bytes lidos indica que não houve *file pruning*. O plano físico da consulta indicou que houve menos *shuffling* e espera para transferência de dados comparado com a carga simples.

Na carga padrão, a tabela fato foi dividida pelo Athena em 12 arquivos de aproximadamente 100MB, já na carga incremental foram gerados 100 arquivos de 22MB (a diferença de volumetria é devido à compressão dos arquivos). Dessa forma, os arquivos menores foram melhor distribuídos entre diferentes nós do cluster, indicando que a configuração padrão de 100MB por arquivo não é ideal para *datasets* com a volumetria apresentada.

Na Tabela 8 são apresentados os valores agregados tratados pelo teste t.

Tabela 8: Performance Relativa entre Hive e Iceberg com carga incremental e SF10.

| Métrica | PR | IC95% | valor p | Melhor performance |
|---|---|---|---|---|
| **Tempo de processamento** | 2.12 | [1.97, 2.28] | <0.001 | Hive |
| **Bytes lidos** | 0.99 | [0.98, 1.00] | 0.10 | Performance igual |

Essa melhora no tempo comparado à carga padrão, porém, vem ao custo do aumento dos bytes lidos em aproximadamente 5% para ambos os casos. No caso do Amazon Athena, isso resulta em um aumento dos custos, apesar de uma diminuição no tempo.

### 5.2.2 SF100

O cenário com *dataset* maior revelou resultados parecidos, com o Hive levando quase metade do tempo do Iceberg. A comparação com a carga padrão também revelou um aumento de performance de 15% para o Hive e 8% para o Iceberg.

O uso da carga incremental no S100 teve menos impacto que no S10. Analisando a divisão dos arquivos, foi evidenciado que a carga padrão já apresentava 30 arquivos para a tabela fato, tornando a separação em 100 menos impactante.

Tabela 9: Média das métricas de execução para SF100 com carga incremental

| Query | Hive | | | Iceberg | | |
|---|---|---|---|---|---|---|
| | T. Proces.(s) | Bytes Lidos(MB) | T. Planej.(s) | T. Proces.(s) | Bytes Lidos(MB) | T. Planej.(s) |
| **Q1.1** | 2.78 | 4373.96 | 0.40 | 5.12 | 4468.87 | 0.37 |
| **Q1.2** | 2.83 | 4373.96 | 0.43 | 5.16 | 4468.87 | 0.38 |
| **Q1.3** | 2.80 | 4373.96 | 0.40 | 4.58 | 4468.87 | 0.30 |
| **Q2.1** | 4.66 | 6395.44 | 0.53 | 7.24 | 6378.30 | 0.50 |
| **Q2.2** | 3.96 | 6395.46 | 0.51 | 6.91 | 6378.30 | 0.53 |
| **Q2.3** | 3.83 | 6395.46 | 0.52 | 6.62 | 6378.30 | 0.52 |
| **Q3.1** | 4.55 | 5564.93 | 0.62 | 7.73 | 5581.21 | 0.61 |
| **Q3.2** | 4.11 | 5568.11 | 0.51 | 7.00 | 5446.60 | 0.64 |
| **Q3.3** | 3.89 | 5508.12 | 0.49 | 6.70 | 5395.88 | 0.52 |
| **Q3.4** | 3.81 | 5555.92 | 0.52 | 6.64 | 5623.77 | 0.51 |
| **Q4.1** | 4.97 | 8229.70 | 0.57 | 8.52 | 8223.25 | 0.73 |
| **Q4.2** | 4.95 | 8692.13 | 0.62 | 8.65 | 8736.41 | 0.71 |
| **Q4.3** | 4.85 | 8694.00 | 0.59 | 8.02 | 8738.30 | 0.76 |
| **Total** | 51.99 | 80121.15 | 6.71 | 88.89 | 80286.93 | 7.08 |

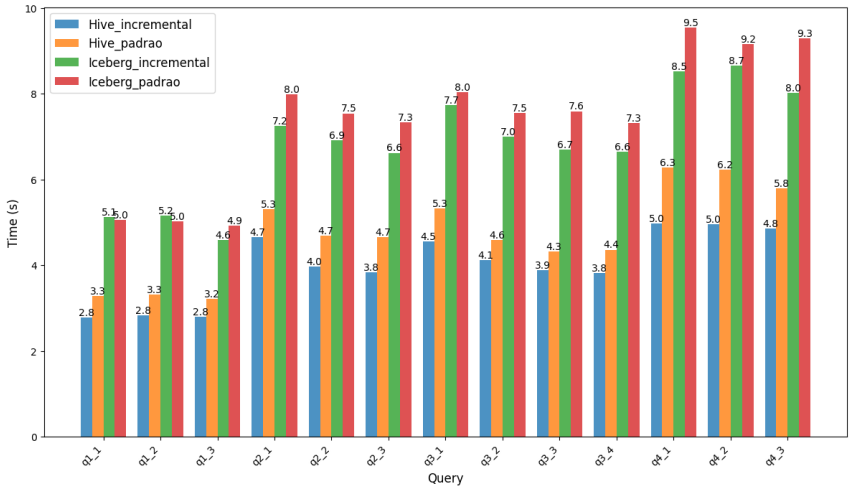


Figura 11: Comparação das cargas incremental e padrão no SF100.

Diferente da carga incremental no SF10, que gerou um aumento nos bytes lidos, nesse caso não houve aumento. Isso gerou um aumento significativo de performance

no tempo com o mesmo custo. Esses resultados reforçam a necessidade de otimizar o tamanho dos arquivos para uma melhor performance analítica.

Tabela 10: Performance Relativa entre Hive e Iceberg com carga incremental e SF100.

| Métrica | PR | IC95% | valor p | Melhor performance |
|---|---|---|---|---|
| **Tempo de processamento** | 1.71 | [1.67, 1.76] | <0.001 | Hive |
| **Bytes lidos** | 1.00 | [0.99, 1.01] | 0.45 | Performance igual |

### 5.2.3 Resultados com OPTIMIZE

O uso da função de optimize, ao compactar os arquivos menores, resultou numa performance intermediária entre a carga padrão e a carga incremental. Apesar disso, como o resultado incremental foi o melhor na performance do tempo, o OPTIMIZE gerou uma latência maior. Na Tabela 11 são comparadas as métricas para as cargas no Iceberg.

Tabela 11: Comparação da operação OPTIMIZE no SF10

| Query | Carga padrão | | Carga Incremental | | OPTIMIZE | |
|---|---|---|---|---|---|---|
| | T. Proc.(s) | Bytes Lidos(MB) | T. Proc.(s) | Bytes Lidos(MB) | T. Proc.(s) | Bytes Lidos(MB) |
| **Q1.1** | 4.33 | 446.69 | 4.02 | 448.14 | 3.47 | 446.81 |
| **Q1.2** | 3.88 | 446.69 | 4.12 | 448.14 | 3.75 | 446.81 |
| **Q1.3** | 3.85 | 446.69 | 4.07 | 448.14 | 3.64 | 446.81 |
| **Q2.1** | 6.40 | 676.69 | 5.22 | 674.17 | 5.08 | 663.28 |
| **Q2.2** | 5.82 | 639.55 | 4.96 | 669.19 | 5.01 | 658.10 |
| **Q2.3** | 5.58 | 641.55 | 4.91 | 654.23 | 4.79 | 638.52 |
| **Q3.1** | 6.28 | 580.94 | 5.33 | 635.87 | 5.02 | 591.66 |
| **Q3.2** | 5.93 | 577.24 | 5.60 | 624.53 | 5.09 | 588.45 |
| **Q3.3** | 5.39 | 572.12 | 5.10 | 626.81 | 5.17 | 589.85 |
| **Q3.4** | 5.35 | 593.49 | 4.80 | 637.22 | 4.56 | 599.22 |
| **Q4.1** | 12.03 | 855.65 | 8.91 | 894.26 | 9.51 | 854.94 |
| **Q4.2** | 10.28 | 850.36 | 7.28 | 885.94 | 8.46 | 855.21 |
| **Q4.3** | 10.30 | 833.05 | 7.02 | 883.56 | 7.76 | 868.15 |
| **Total** | 85.42 | 8160.71 | 71.34 | 8530.2 | 71.31 | 8247.81 |

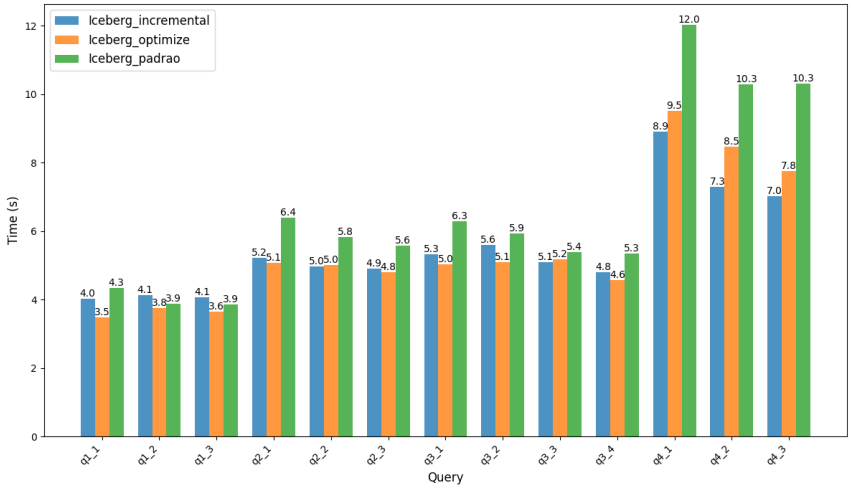


Figura 12: Comparação das cargas incremental, padrão e optimize para Iceberg no SF10.

A quantidade de bytes lidos também ficou entre a carga padrão e a incremental, mas, favorecendo o uso do OPTIMIZE, já que reduz o custo de leitura mas mantém o tempo. O resultado mostra como essa funcionalidade pode ser útil no gerenciamento da tabela.

A operação para o SF100 apresentou os mesmos resultados que o SF10, porém de forma menos expressiva, dada a menor diferença entre a carga incremental e a padrão.

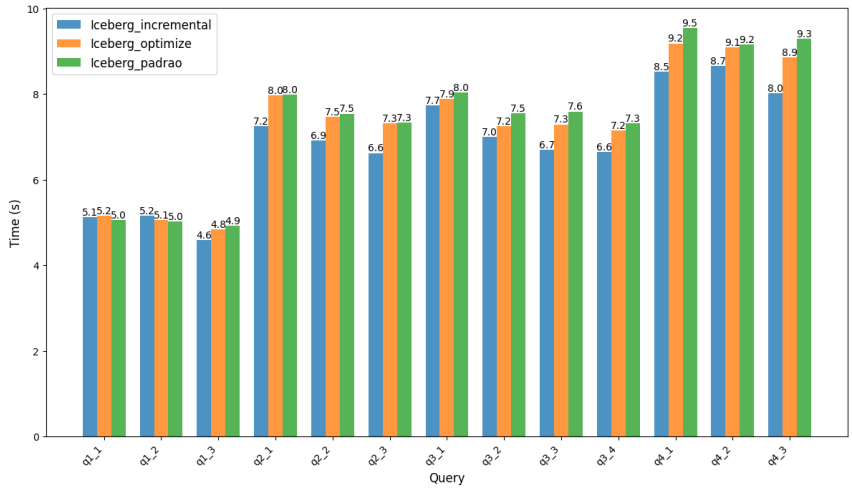


Figura 13: Comparação das cargas incremental, padrão e optimize para Iceberg no SF100.

Foi aplicado o conceito da Performance Relativa para verificar se a utilização do OPTIMIZE gerou benefícios após a carga incremental. Os resultados do teste t não foram totalmente conclusivos e são exibidos na Tabela 12.

Tabela 12: Performance Relativa entre Iceberg com carga incremental e OPTIMIZE.

| Métrica | SF | PR | IC95% | valor p | Melhor performance |
|---|---|---|---|---|---|
| **Tempo de processamento** | SF10 | 1.10 | [1.05, 1.14] | <0.001 | OPTIMIZE |
| **Bytes lidos** | SF10 | 1.03 | [1.02, 1.05] | <0.001 | OPTIMIZE |
| **Tempo de processamento** | SF100 | 0.96 | [0.94, 0.98] | <0.001 | Incremental |
| **Bytes lidos** | SF100 | 1.00 | [1.00, 1.01] | 0.002 | Performance igual |

## 5.3 **Resultados divergentes com o EXPLAIN ANALYZE**

A função EXPLAIN ANALYZE foi utilizada no Amazon Athena para entender melhor a execução das consultas e gerar insights. Foi notado, contudo, uma grande diferença na performance das consultas com e sem a ferramenta. Rodando as consultas através dessa ferramenta, notou-se que os resultados do Iceberg e Hive ficaram bem mais próximos.

Uma possível explicação são checagens de metadados que o Athena faz na tabela Iceberg ao rodar uma *query*. No modo de operação Merge-On-Read do Iceberg no Athena, uma consulta de leitura pode gerar mudanças nos arquivos de dados e gerar uma maior latência. Porém, isso só deveria ocorrer uma vez após a escrita e não foram notadas influências dessa funcionalidade nos dados das primeiras consultas.

Embora existam outros artigos comprovando a disparidade de performance na leitura entre Iceberg e Hive, a explicação não foi explorada [44]. O uso do Athena para fazer essas explorações também não entregou respostas conclusivas. Entre sua estrutura serverless, a falta de recursos de logging e metrificação adequados e as diferenças inexplicáveis nas ferramentas de debug indicam que o Athena, embora amplamente utilizado, não é adequado para realização de testes aprofundados.

# 6 CONCLUSÃO E TRABALHOS FUTUROS

## 6.1 Conclusão

Este estudo apresentou uma série de resultados analisando o uso comparativo das tabelas Hive e Iceberg. Os dados aqui expostos podem ser, então, utilizados por arquitetos e engenheiros de dados para melhor embasar suas decisões em um ambiente de Data Lakehouse.

O Iceberg apresentou uma performance bem abaixo de uma tabela Hive em termos de tempo de processamento. Contudo, em termos de custo, ambas as soluções apresentaram o mesmo desempenho, já que a leitura de dados foi a mesma. A investigação sobre os motivos dessa diferença não foi conclusiva devido às poucas ferramentas providas pela *engine*.

O Athena apresentou grande variabilidade no tempo de execução das consultas, resultando em grandes desvios-padrão, especialmente para o Iceberg. Embora esse comportamento seja esperado em um serviço serverless que não cobra pelo tempo, a imprevisibilidade dificulta a melhor compreensão e otimização das operações.

O tamanho dos arquivos mostrou ser uma variável importante para a velocidade do processamento. Embora arquivos muito pequenos levem a problemas de performance, foi notado que o tamanho padrão para arquivos escritos pelo Athena não estava otimizado. O uso de estratégias de *bucketing* e ajustes na configuração das tabelas podem ser aliados nesse sentido.

Estratégias de VACUUM e OPTIMIZE mostraram trazer os resultados mais próximos de uma tabela com carga única. Porém, os testes com essa ferramenta não foram suficientes para validar seu uso, já que não houve fragmentação suficiente para gerar perda de performance. Investigações mais avançadas com operações *row-wise*, uso de updates e particionamento podem gerar um cenário mais rico para a exploração dessa funcionalidade.

Ainda que o Iceberg tenha apresentado resultados inferiores, ele ainda se apresenta como uma forte ferramenta no mundo do DLH. A gama de funcionalidades providas vão muito além da performance analítica em consultas simples. As melhorias na implementação de partições podem ter altíssimo impacto em vários cenários de consultas que não foram investigados neste trabalho. Além disso, a possibilidade de

transações ACID, evolução de esquema e updates são grandes atrativos para a adoção dessa plataforma.

## 6.2 **Trabalhos futuros**

Ainda existem muitos tópicos para pesquisa e investigação sobre o Apache Iceberg e a área de Data Lakehouse como um todo. Trabalhos futuros podem investigar alternativas não exploradas no presente trabalho como:

- Explorar o uso de partições e técnicas de *bucketing*.
- Testes em outras *query engines* como Trino e Spark.
- Utilizar ambientes de computação com recursos dedicados.
- Testes com diferentes *metastores* e suas configurações.
- Variações de cargas de trabalho com mudanças de esquema e operações de update.
- Otimizações dos parâmetros das tabelas como tamanho ideal para arquivos de dado.

# REFERÊNCIAS


[1] ARMBRUST, M. et al. Lakehouse: a new generation of open platforms that unify data warehousing and advanced analytics. In: *Proceedings of CIDR*. [S.l.: s.n.], 2021. v. 8, p. 28.

[2] SAHA, D. Disruptor in data engineering-comprehensive review of apache iceberg. *Available at SSRN 4987315*, 2024.

[3] APACHE Hive. Disponível em: `http://hive.apache.org/`. Acesso em: 20 mar. 2025.

[4] O'NEIL, P. et al. The star schema benchmark. *University of Massachusetts, Boston, Tech. Rep*, 2007.

[5] MAYER-SCHÖNBERGER, V.; CUKIER, K. *Big data: A revolution that will transform how we live, work, and think*. [S.l.]: Houghton Mifflin Harcourt, 2013.

[6] CHEN, M.; MAO, S.; LIU, Y. Big data: A survey. *Mobile networks and applications*, Springer, v. 19, p. 171–209, 2014.

[7] ANURADHA, J. et al. A brief introduction on big data 5vs characteristics and hadoop technology. *Procedia computer science*, Elsevier, v. 48, p. 319–324, 2015.

[8] APACHE Hadoop Wiki. Disponível em: `https://cwiki.apache.org/confluence/display/HADOOP2`. Acesso em: 20 mar. 2025.

[9] VOHRA, D. Practical hadoop ecosystem. *Chapter in Apache Parquet*, Springer, v. 177, p. 178, 2016.

[10] HASHEM, I. A. T. et al. The rise of "big data" on cloud computing: Review and open research issues. *Information Systems*, v. 47, p. 98–115, 2015. ISSN 0306-4379. Available at: <https://www.sciencedirect.com/science/article/pii/S0306437914001288>.

[11] AMAZON EMR. Disponível em: `https://aws.amazon.com/pt/emr/`. Acesso em: 20 mar. 2025.

[12] DATABRICKS. Disponível em: https://www.databricks.com/br. Acesso em: 20 mar. 2025.

[13] DATAPROC. Disponível em: https://cloud.google.com/dataproc. Acesso em: 20 mar. 2025.

[14] FACTOR, M. et al. Object storage: The future building block for storage systems. In: . [S.l.: s.n.], 2005. p. 119 – 123. ISBN 0-7803-9228-0.

[15] AMAZON S3. Disponível em: https://aws.amazon.com/s3/. Acesso em: 20 mar. 2025.

[16] GOOGLE Cloud Storage. Disponível em: https://cloud.google.com/storage. Acesso em: 20 mar. 2025.

[17] AZURE Blob Storage. Disponível em: https://azure.microsoft.com/en-us/products/storage/blobs. Acesso em: 20 mar. 2025.

[18] NAMBIAR, A.; MUNDRA, D. An overview of data warehouse and data lake in modern enterprise data management. *Big Data and Cognitive Computing*, v. 6, n. 4, 2022. ISSN 2504-2289. Available at: <https://www.mdpi.com/2504-2289/6/4/132>.

[19] INMON, W. H. *Building the data warehouse*. [S.l.]: Wiley, 2005.

[20] AMAZON Redshift. Disponível em: https://aws.amazon.com/pt/redshift/. Acesso em: 20 mar. 2025.

[21] GOOGLE Big Query. Disponível em: https://cloud.google.com/bigquery. Acesso em: 20 mar. 2025.

[22] DIXON, J. Pentaho, hadoop, and data lakes. 2010. Available at: <https://jamesdixon.wordpress.com/2010/10/14/pentaho-hadoop-and-data-lakes/>.

[23] THUSOO, A.; AL., J. S. S. et. Hive - a Petabyte Scale Data Warehouse using Hadoop. *ICDE*, p. 996–1005, 2010.

[24] OREščANIN, D.; HLUPIć, T. Data lakehouse - a novel step in analytics architecture. In: *2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO)*. [S.l.: s.n.], 2021. p. 1242–1246.

[25] APACHE Hudi. Disponível em: https://hudi.apache.org/. Acesso em: 20 mar. 2025.

[26] APACHE Iceberg. Disponível em: https://iceberg.apache.org/. Acesso em: 20 mar. 2025.

[27] DELTA Lake. Disponível em: https://delta.io/. Acesso em: 20 mar. 2025.

[28] CLARK, L. *Apache Iceberg promises to change the economics of cloud-based data analytics*. Disponível em: https://www.theregister.com/2023/01/03/apache_iceberg/. Acesso em: 20 mar. 2025.

[29] SADLER, S.; PAREKH, R.; MALKANI., A. *Iceberg at Adobe*. Disponível em: https://blog.developer.adobe.com/iceberg-at-adobe-88cf1950e866. Acesso em: 20 mar. 2025.

[30] SPITZER, R. *Iceberg at Apple*. Disponível em: https://www.dremio.com/subsurface/on-demand/iceberg-development-at-apple/. Acesso em: 20 mar. 2025.

[31] CLARK, L. *Databricks puts cards on the table format as Snowflake looks for more players*. Disponível em: https://www.theregister.com/2023/06/29/databricks_snowflake_tables/. Acesso em: 20 mar. 2025.

[32] APACHE Iceberg Spec. Disponível em: https://iceberg.apache.org/spec/. Acesso em: 20 mar. 2025.

[33] HARTZELL, K. Comparison of big data sql engines in the cloud. Helsingin yliopisto, 2023.

[34] SETHI, R. et al. Presto: Sql on everything. In: IEEE. *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. [S.l.], 2019. p. 1802–1813.

[35] TRINO partition pruning. Disponível em: https://trino.io/blog/2020/06/14/dynamic-partition-pruning.html. Acesso em: 20 mar. 2025.

[36] KULKARNI, A. Amazon athena: Serverless architecture and troubleshooting. *International Journal of Computer Trends and Technology*, v. 71, n. 5, p. 57–61, 2023.

[37] AMAZON Athena. Disponível em: `https://aws.amazon.com/athena/`. Acesso em: 20 mar. 2025.

[38] SAXENA, M. et al. The story of aws glue. 2023.

[39] POESS, M.; FLOYD, C. New tpc benchmarks for decision support and web commerce. *SIGMOD Rec.*, Association for Computing Machinery, New York, NY, USA, v. 29, n. 4, p. 64–71, dez. 2000. ISSN 0163-5808. Available at: <https://doi.org/10.1145/369275.369291>.

[40] POESS, M. et al. Tpc-ds, taking decision support benchmarking to the next level. In: *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: Association for Computing Machinery, 2002. (SIGMOD '02), p. 582–587. ISBN 1581134975. Available at: <https://doi.org/10.1145/564691.564759>.

[41] ADAMSON, C. *Mastering data warehouse aggregates: solutions for star schema performance*. [S.l.]: John Wiley & Sons, 2012.

[42] VARGAS, R. F. L. A performance comparison of data lake table formats in cloud object storages. 2022.

[43] FERREIRA, F. E. R. R.; FIDALGO, R. do N. A performance analysis of hybrid and columnar cloud databases for efficient schema design in distributed data warehouse as a service. *Data*, MDPI, v. 9, n. 8, p. 99, 2024.

[44] NELLURI, S. R.; SALDANHA, F. A. A. Mastering big data formats: Orc, parquet, avro, iceberg, and the strategy of selection. *International Journal of Computer Trends and Technology*, v. 73, p. 44–50, 2025.

[45] DUCKDB. Disponível em: `https://duckdb.org/`. Acesso em: 20 mar. 2025.

[46] LABORãO, E. *Scripts utilizados nos experimentos*. Disponível em: `https://github.com/Enriqson/tcc/`.

[47] RAHMAN, M. M.; HASAN, M. H. Serverless architecture for big data analytics. In: IEEE. *2019 Global Conference for Advancement in Technology (GCAT)*. [S.l.], 2019. p. 1–5.

[48] APACHE Parquet. Disponível em: https://parquet.apache.org/. Acesso em: 20 mar. 2025.