Centro de Informática
UFPE

Nicholas Henrique Justino Ferreira

Uma Análise de Desempenho dos Clusters do SingleStoreDB Cloud Comparando o Armazenamento RowStore e ColumnStore em Banco de Dados do Tipo Data Warehouse

Universidade Federal de Pernambuco
coord-ec@cin.ufpe.br
https://portal.cin.ufpe.br/graduacao/

Recife
2023

Nicholas Henrique Justino Ferreira

Uma Análise de Desempenho dos Clusters do SingleStoreDB Cloud Comparando o Armazenamento RowStore e ColumnStore em Banco de Dados do Tipo Data Warehouse

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

Área de Concentração: Banco de Dados
Orientador: Prof. Dr. Robson do Nascimento Fidalgo

Recife
2023

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

Ferreira, Nicholas Henrique Justino.
Uma análise de desempenho dos clusters do SingleStoreDB Cloud comparando o armazenamento RowStore e ColumnStore em banco de dados do tipo Data Warehouse / Nicholas Henrique Justino Ferreira. - Recife, 2023.
48 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, 2023.
Inclui referências, apêndices.

1. NewSQL. 2. DBaaS. 3. SSB. 4. Data Warehouse. I. Fidalgo, Robson do Nascimento. (Orientação). II. Título.

000 CDD (22.ed.)

BANCA

Dedico este Trabalho de Conclusão de Curso à toda minha família, que sempre me deram todo o suporte e apoio para que eu pudesse concluir a minha graduação.

AGRADECIMENTOS

Agradeço primeiramente ao meu pai, Jeferson, que possibilitou que eu começasse e continuasse a minha graduação neste curso e universidade até a minha presente conclusão.

Agradeço à toda minha família, amigos e colegas que estiveram presentes nesta jornada da minha graduação e me ajudaram a superar os desafios e obstáculos que enfrentei.

Agradeço ao meu orientador, Prof. Dr. Robson do Nascimento Fidalgo, por ter aceito me orientar neste trabalho e ter se mantido disposto a mostrar o caminho para a conclusão deste trabalho. E não menos importante, agradeço ao Fred Rabelo por toda a consultoria e apoio dado a mim durante o desenvolvimento da pesquisa.

RESUMO

Data Warehouses são Bancos de Dados históricos de uma determinada área com a finalidade de permitir realizar análises para tomadas de decisão. Devido ao grande volume de dados que um Data Warehouse armazena, eles se tornaram um item essencial para empresas que buscam agregar conhecimento com base em seus dados para traçar estratégias de crescimento e melhoria. O NewSQL é um novo paradigma de Sistemas de Gerenciamento de Banco de Dados que visa unir o melhor dos Bancos de Dados Relacionais e NoSQL, proporcionando operações Online Transactional Processing e Online Analytical Processing. Neste trabalho, é feito o escalonamento vertical de 5 *clusters* do SingleStoreDB Cloud, um Database as a Service NewSQL, para uma análise de desempenho dos formatos de armazenamento *RowStore* e *ColumnStore*, com base no Star Schema Benchmark, um *benchmark* para Data Warehouse. Esta análise é feita para as métricas de tempo de execução na Central Unit Processing, tempo total decorrido de consulta e consumo de Random Access Memory. Os resultados mostram que o *RowStore* executa as consultas mais rápido na Central Unit Processing do que o *ColumnStore* tendo desempenhos semelhantes em todos os *clusters*. Mas, o *RowStore* necessita de mais tempo total de consulta do que o *ColumnStore*, onde não foi percebido uma correspondência do escalonamento dos *clusters* com o valor da métrica. E, por fim, o *ColumnStore* necessitou de mais Random Access Memory do que o *RowStore*, no qual novamente não teve uma relação entre o escalonamento e a métrica. Também é mostrado o volume de dados necessário para o armazenamento em *RowStore* e *ColumnStore*.

Palavras-chave: NewSQL. DBaaS. SSB. Data Warehouse. RowStore. ColumnStore. OLAP. OLTP.

ABSTRACT

Data Warehouses are historical databases of a given area in order to allow analysis for decision making. Due to the large volume of data that a Data Warehouse stores, they have become an essential item for companies that seek to add knowledge based on their data to outline strategies for growth and improvement. NewSQL is a new paradigm of Database Management Systems that aims to unite the best of Relational and NoSQL Databases, providing Online Transactional Processing and Online Analytical Processing operations. In this work, 5 clusters of SingleStoreDB Cloud, a Database as a Service NewSQL, are vertically scaled for a performance analysis of the RowStore and ColumnStore storage formats, based on the Star Schema Benchmark, a benchmark for Data Warehouse. This analysis is done for the Central Unit Processing execution time, total elapsed query time and Random Access Memory consumption metrics. The results show that the RowStore executes queries faster in the Central Unit Processing than the ColumnStore, having similar performances in all clusters. But, the RowStore needs more total query time than the ColumnStore, where a correspondence between the scaling of the clusters and the metric value was not noticed. And finally, ColumnStore required more Random Access Memory than RowStore, which again did not have a relationship between scaling and metric. Also shown is the volume of data required for storage in RowStore and ColumnStore.

Keywords: NewSQL. DBaaS. SSB. Data Warehouse. RowStore. ColumnStore. OLAP. OLTP.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

ACID	Atomicidade, Consistência, Isolamento e Durabilidade
BD	Banco de Dados
BDs	Bancos de Dados
CPU	Central Processing Unit
DBaaS	Database as a Service
DDL	Data Definition Language
DML	Data Manipulation Language
DW	Data Warehouse
DWs	Data Warehouses
GB	Gigabyte
GBs	Gigabytes
HD	Hard Drive
HTAP	Hybrid Transactional Analytical Processing
MBs	Megabytes
NoSQL	Not Only SQL
OLAP	*Online Analytical Processing*
OLTP	Online Transactional Processing
RAM	Random Access Memory
SF	Scale Factor
SFs	Scale Factors
SGBD	Sistema de Gerenciamento de Banco de Dados
SGBDs	Sistemas de Gerenciamento de Banco de Dados
SQL	Structured Query Language
SSB	Star Schema Benchmark

SUMÁRIO

1 INTRODUÇÃO

Desde os primórdios do universo, após o *Big Bang*, o cosmos está constantemente produzindo um número inimaginável de dados em diversas formas, formatos e velocidades. Tais dados, podem ser captados, tratados e modelados para nos entregar valor, a informação, que nos fez e nos faz progredirmos na ciência, como seres humanos e em relações interpessoais. Para ser possível o estudo e análise desses dados, foi necessário a criação de Sistemas de Gerenciamento de Banco de Dados (SGBDs) que fossem capazes de armazenar essas informações para seu uso e posterior análise.

Neste capítulo, serão apresentados os temas introdutórios deste trabalho. Na Seção 1.1, é apresentado uma contextualização sobre o escopo deste trabalho. Na Seção 1.2, encontra-se o motivo que levou a realização desta pesquisa. Na Seção 1.3, é evidenciado o objetivo que o trabalho possui para a verificação da motivação. Por fim, na Seção 1.4, é mostrada como é a divisão deste trabalho.

1.1 CONTEXTO

Entre os Bancos de Dados (BDs) mais populares e utilizados no mundo, destacam-se os BDs Relacionais e Not Only SQL (NoSQL). Edgar F. Codd apresentou seu trabalho sobre BDs relacionais (Codd, 1970), onde a relação dos dados foi baseada na Teoria dos Conjuntos. Tal modelo de Banco de Dados (BD) se tornou o mais popular e utilizado até os dias de hoje, devido a padronização de sua linguagem de consulta e o uso de dados estruturados. Com o advento da *Internet*, novos tipos de dados se popularizaram, como as imagens, vídeos e representações de relacionamento entre entidades, que são classificados como semiestruturados e não estruturados. Tais dados não são melhores representados em um modelo relacional, motivando o desenvolvimento de BDs NoSQL (Han *et al.*, 2011; Strauch, 2011), os quais proveem uma alta disponibilidade e rapidez quando comparada aos BDs relacionais, mas diferentemente destes não po.

Com o objetivo de superar as limitações dos BDs Relacionais e NoSQL, foi criado um novo tipo de BD, o NewSQL. Diferentemente dos BDs Relacionais, os BDs NewSQL fornecem escalabilidade horizontal, alta disponibilidade sem comprometer a consistência e lidam com uma grande quantidade de dados. Por sua vez, com relação aos BDs NoSQL, os BDs

NewSQL atendem à necessidade de transações com propriedades de Atomicidade, Consistência, Isolamento e Durabilidade (ACID) e linguagens Structured Query Language (SQL). Em resumo, os bancos de dados NewSQL preenchem a lacuna entre os BDs relacionais tradicionais e os BDs NoSQL modernos, oferecendo o melhor dos dois mundos (Pavlo & Aslett, 2016).

Quando se trata de tipos de armazenamento, um Data Warehouse (DW) provê um repositório de dados históricos capaz de entregar informações para uma tomada de decisão para as empresas e corporações. Sua implementação ocorre normalmente em BDs Relacionais, mas devido a possibilidade de gerenciar imensos volumes de dados, os BDs Relacionais se tornam alternativas caras e até inviáveis para processar e analisar, com bom desempenho, esses repositórios de dados (Stonebraker *et al.*, 2007).

1.2 MOTIVAÇÃO

O surgimento de SGBDs no modelo Database as a Service (DBaaS) (Muhammed *et al.*, 2021) foi uma grande revolução no mercado, pois permitiu que os usuários não necessitassem mais manter seus próprios servidores e máquinas para acomodar seus BDs. Estas responsabilidades foram atribuídas para empresas especializadas em prover serviços na *Internet*, através de um provedor de computação distribuído em diversos servidores espalhados pelo mundo. Tal provedor de computação, chamado de *Cloud Computing*, gerencia, armazena e mantêm todos os recursos de computação (servidores, *software*, armazenamento e rede) e permite ao usuário o acesso a qualquer momento e o seu uso de forma escalonável e flexível (Kim, 2009).

Portanto, devido a sua inovação, existe a necessidade de uma avaliação de desempenho do escalonamento vertical dos *clusters* de um Sistema de Gerenciamento de Banco de Dados (SGBD) NewSQL no modelo DBaaS para analisar quais são os melhores cenários de uso dos formatos de armazenamento *RowStore* e *ColumnStore* para um DW.

1.3 OBJETIVO

O estudo deste trabalho visa a análise dos *clusters* do SGBD SingleStoreDB Cloud, para avaliar as características de desempenho entre *RowStore* e *ColumnStore* para um DW simulado pelo Star Schema Benchmark (SSB) em *clusters* do novo paradigma de SGBDs NewSQL. Para isso, realiza-se o escalonamento vertical dos *clusters* do SGBD SingleStore e avalia-se as métricas de desempenho do SGBD, em relação ao tempo de uso de Central Processing Unit (CPU), tempo total de consulta decorrido, quantidade de memória Random Access Memory (RAM) utilizada e volume necessário para o armazenamento do SSB.

1.4 ORGANIZAÇÃO DO TRABALHO

Este trabalho está constituído de 5 capítulos: no Capítulo 1, é apresentado uma visão geral sobre a área de estudo desta pesquisa, assim como, a sua motivação e o seu objetivo; No Capítulo 2, são apresentados os conceitos relacionados ao desenvolvimento do trabalho; O Capítulo 3, apresenta a metodologia utilizada para realizar os experimentos; No Capítulo 4, os resultados do Capítulo 3 são apresentados e explicados; Por fim, no Capítulo 5, apresenta-se a conclusão do estudo deste trabalho e futuros estudos relacionados à pesquisa.

2

CONCEITOS

Neste capítulo, serão apresentados os temas que englobam os conceitos deste trabalho: a Seção 2.1, será descrito o tipo de BD NewSQL; Na Seção 2.2, será feita uma análise comparativa entre os modelos de armazenamento *RowStore* e *ColumnStore*; Na Seção 2.3, será apresentado os conceitos, características e modelagem de um DW; E, por fim, na Seção 2.4, será abordado o *benchmark* de DW SSB.

2.1 NEWSQL

NewSQL é um termo usado para descrever uma categoria de SGBDs que visa fornecer os benefícios de um SGBD Relacional e NoSQL, ao mesmo tempo que visa superar suas desvantagens. Esses bancos de dados foram projetados para lidar com as demandas de escalabilidade horizontal de aplicativos de Big Data e em nuvem, sem comprometer a consistência dos dados. O objetivo principal do NewSQL é fornecer o desempenho, escalabilidade e confiabilidade necessários para aplicativos críticos de negócios.

O NewSQL agrupa a escalabilidade e desempenho de SGBDs NoSQL com a consistência e confiabilidade de SGBDs Relacionais. Eles compõem uma moderna classe de SGBD que atende a necessidade de um SGBD mais escalável e de alto desempenho sem sacrificar as propriedades ACID de SGBDs Relacionais (Kaur & Sachdeva, 2017; Pavlo & Aslett, 2016). Adicionalmente, o NewSQL oferece tolerância a falhas e um grande número de transações em vários nós simultaneamente sem bloqueio de controle (Muhammed *et al.*, 2021).

A arquitetura do NewSQL além de permitir a escalabilidade horizontal, também admite escalabilidade vertical, e sua escalabilidade horizontal é fornecida pelo emprego de particionamento, replicação e *clustering* para que as consultas não necessitem se comunicar entre muitos nós (Binani *et al.*, 2016; Moniruzzaman, 2014). Porém, o NewSQL apresenta as desvantagens de: ser pouco adotado, o que limita a disponibilidade e suporte a eles; poder ser complexo de gerir e configurar, necessitando de uma grande capacidade técnica. Todavia, o NewSQL representa uma nova direção promissora na tecnologia de SGBDs que oferece uma alternativa atraente aos SGBDs Relacionais e NoSQL, ao combinar os melhores recursos desses dois tipos de SGBDs.

2.2 ROWSTORE VS. COLUMNSTORE

Os SGBDs têm sido amplamente utilizados para armazenar e gerenciar dados em vários domínios. Dentro desse contexto, duas arquiteturas de armazenamento têm sido amplamente discutidas: o *RowStore* e o *ColumnStore*. O *RowStore* é a arquitetura tradicional de armazenamento, na qual os dados são armazenados linha por linha de maneira sequencial na memória secundária. Já a arquitetura *ColumnStore*, por sua vez, armazena as informações dos dados na memória secundária coluna a coluna.

A escolha entre o uso do *RowStore* ou do *ColumnStore* pode ser feita dependendo do tipo de aplicação a ser executada. O *RowStore* é mais indicado para operações Online Transactional Processing (OLTP), enquanto o *ColumnStore* é mais apropriado para operações *Online Analytical Processing* (OLAP). As operações OLTP envolvem transações pequenas, mas frequentes, que exigem a leitura ou gravação de linhas inteiras de dados. Já as operações OLAP envolvem a análise de grandes volumes de dados.

As vantagens da arquitetura *ColumnStore* residem na capacidade de melhorar o desempenho de consultas para um grande conjunto de dados e na compactação dos dados, pois é esperado que os dados em uma mesma coluna sejam do mesmo tipo, o que facilita a compressão do dado. A compressão também é atingida quando há dados consecutivos repetidos, que são sumarizados na memória (Abadi *et al.*, 2008). Além disso, a arquitetura *ColumnStore* é ideal para consultas analíticas, pois se beneficia do armazenamento em prol da seletividade das colunas em suas operações (Sterjo, 2017).

Por outro lado, o *RowStore* é mais adequado para aplicações OLTP, nas quais as transações são pequenas e envolvem a leitura ou gravação de algumas linhas de dados. Nesse caso, a arquitetura *RowStore* é mais eficiente do que a *ColumnStore*, pois permite um acesso mais rápido aos dados de uma determinada linha. No entanto, o *RowStore* não é eficiente para consultas analíticas, que envolvem a análise de grandes volumes de dados.

ColumnStore é uma técnica frequentemente utilizada em *Data Warehouses* para o armazenamento dos dados, devido ao seu desempenho em selecionar colunas específicas para realizar análises sobre os dados. Por isso, são mais apropriados do que os tradicionais BDs *RowStore* (Abadi *et al.*, 2008; S.Kanade & Gopal, 2013) para as áreas estratégicas, como o *Bussiness Inteligence*. A comparação entre os dois formatos de armazenamento está exemplificada na Figura 1.

Tabela 1: Comparação entre *RowStore* e *ColumnStore*

Características	RowStore	ColumnStore
Armazenamento de dados	Salva os dados linha por linha sequencialmente na memória secundária	Armazena os dados coluna por coluna na memória secundária
Aplicação	Melhor para operações OLTP	Melhor para operações OLAP
Compactação de dados	Não é eficiente, pois os dados são armazenados semelhantes a uma tabela tradicional	Muito eficiente, pois os dados em uma mesma coluna são do mesmo tipo, facilitando a compressão
Aplicação	Aplicação ideal em sistemas de bancos de dados transacionais	Ideal para armazenar grandes volumes de dados em sistemas analíticos

2.3 DATA WAREHOUSE

Um DW é um BD com grandes volumes de dados estruturados, não-voláteis, históricos e direcionados a um tópico, com o intuito de oferecer operações de análises de dados (Kimball & Ross, 2013). O DW contém dados consolidados de muitos anos, podendo ser derivado de mais de um BD e possuir centenas de Gigabytes (GBs) e chegar até a Terabytes de armazenamento.

Por ter um propósito analítico, os Data Warehouses (DWs) são caracterizados por serem BDs do tipo OLAP, portanto, são focados em consultar dados com muitos registros para dar suporte a tomada de decisão através de análises e geração de relatórios, para aplicações de *Business Inteligence*. Os BDs OLAP são otimizados para intensas cargas de trabalho, onde consultas complexas podem acessar milhões de registros e realizar operações multidimensionais, com *joins*, agregações e comparações. Em contrapartida, BDs do tipo OLTP, são designados para processamento transacional, como operações de Create, Read, Update e Delete que visam proporcionar a manutenção de um BD com dados atuais. Portanto, são otimizados para transações de alto volume e de tempo de resposta de baixa latência, dessa maneira, são utilizados em aplicações bancárias, industriais, de *e-commerce*, entre outras.

Um DW possui três tipos de esquema, o esquema estrela (*Star Schema*), o floco de neve (*SnowFlake*) e a constelação (*Fact Constellation*). Neles, existem dois tipos de tabelas, a tabela de fatos e tabela de dimensão. A tabela de fatos é onde reside as informações históricas da área que está sendo analisada em questão, e as tabelas de dimensão possuem informações descritivas complementares.

O *Star Schema* tem a característica de ser dividido em uma tabela de fatos e várias tabelas de dimensão, tendo seus dados nas tabelas de dimensão desnormalizados e sendo o esquema mais utilizado. O *SnowFlake* possui a característica de ser um modelo mais normalizado do que o *Star*

Schema, ou seja, suas tabelas de dimensão são subdivididas em tabelas menores para exercerem ainda mais a função descritiva para outras tabelas. Consequentemente, isto reduz a redundância e o volume de dados armazenados, mas pode necessitar de mais tempo e processamento, pois necessita realizar mais *joins* em suas consultas. Por fim, o *Fact Constellation* é uma coleção de de tabelas de fatos que compartilham o uso das tabelas de dimensão e seu uso é apropriado para situações onde mais de uma área precisa ser analisada e não há uma hierarquia entre as tabelas de fatos.

2.4 STAR SCHEMA BENCHMARK

O SSB é uma ferramenta de *benchmarking* projetada para avaliar o desempenho de BDs em cenários típicos de DW. Ele é constituído de 5 tabelas no esquema estrela, são elas: a tabela de fatos, *lineorder*, e as tabelas de dimensão, *customer*, *supplier*, *part* e *date*. O esquema de estrela é uma abordagem de modelagem de dados comum em DW que consiste em uma tabela de fatos central conectada a várias tabelas de dimensão. A Figura 1 mostra o esquema do SSB, assim como, o esquema de suas tabelas.

O SSB também é composto por 4 Scale Factors (SFs), o SF 1, SF 10, SF 100 e SF 1000, que se diferenciam na quantidade de linhas das tabelas *lineorder*, *customer*, *supplier*, *part* e *date*. Quanto maior o número do SF, maior o número de linhas nessas tabelas, com exceção da tabela *date* que mantêm o mesmo número de linhas para todos os SFs. O objetivo dos SFs é permitir que os usuários comparem o desempenho do SGBD em diferentes cenários de carga de trabalho. Para este estudo, foram escolhidos os SFs 10 e 100, a quantidade de linhas deles se encontram na Tabela 2. O tamanho original dos arquivos em GBs destes SFs são iguais a 5.68 e 58, respectivamente.

Figura 1: Diagrama do Star Schema Benchmark

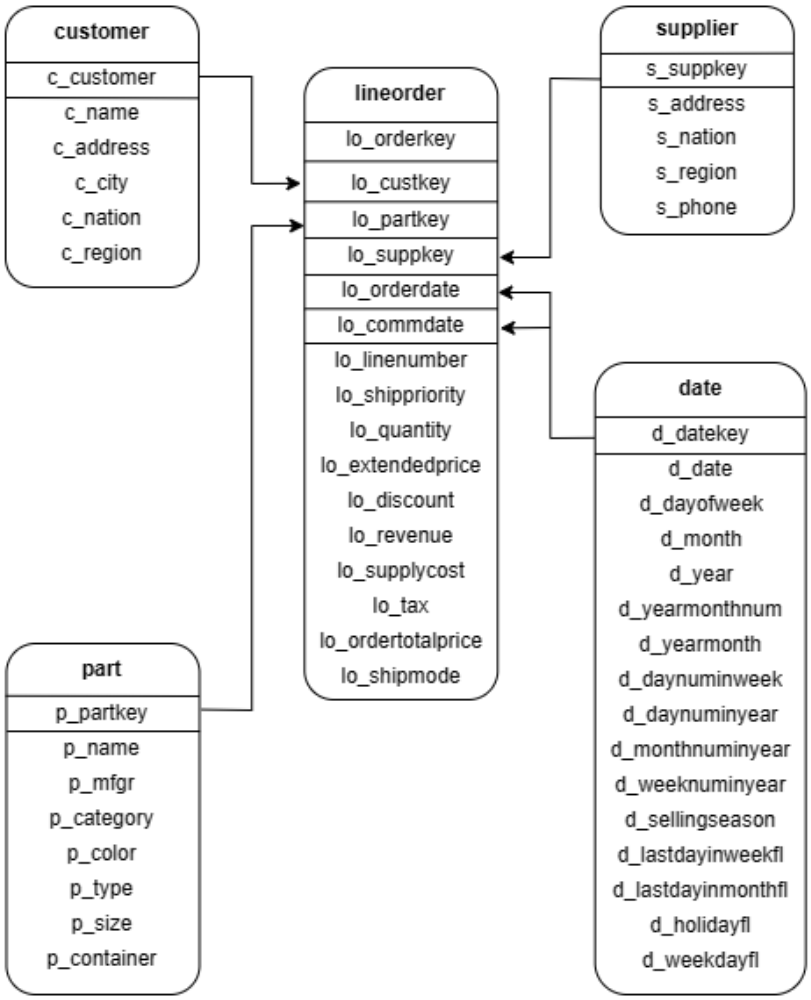


Tabela 2: Quantidade de linhas dos Scale Factors do Star Schema Benchmark

	SF 10	SF 100
lineorder	59986214	600037902
supplier	20000	200000
customer	300000	3000000
part	800000	1400000
date	2556	2556

O SSB possui 13 consultas divididas em 4 grupos que visam medir o desempenho do DW de acordo com o tipo de complexidade que está atribuído ao seu grupo da consulta. O Grupo 1 apresenta a característica de utilizar as tabelas *lineorder* e *date*, o Grupo 2, utiliza as tabelas *lineorder*, *date*, *part* e *supplier*. O Grupo 3 utiliza as tabelas *customer*, *lineroder*, *supplier* e *date*, por fim, o Grupo 4 utiliza as tabelas *date*, *customer*, *supplier*, *part* e *lineorder*. As consultas e seus grupos podem ser visualizados na Tabela 3.

Tabela 3: Classificação das consultas do Star Schema Benchmark

Query	Grupo	ID
SELECT SUM(lo_extendedprice*lo_discount) AS revenue FROM lineorder, dwdate WHERE d_yearmonthnum = 199401 AND lo_orderdate = d_datekey AND lo_discount BETWEEN 4 AND 6 AND lo_quantity BETWEEN 26 AND 35;	1	0
SELECT SUM(lo_extendedprice*lo_discount) AS revenue FROM lineorder, dwdate WHERE d_year = 1993 AND lo_orderdate = d_datekey AND lo_discount BETWEEN 1 AND 3 AND lo_quantity < 25;	1	1
SELECT SUM(lo_extendedprice*lo_discount) AS revenue FROM lineorder, dwdate WHERE d_weeknuminyear = 6 AND lo_orderdate = d_datekey AND d_year = 1994 AND lo_discount BETWEEN 5 AND 7 AND lo_quantity BETWEEN 26 AND 35;	1	2
SELECT SUM(lo_revenue), d_year, p_brand FROM lineorder, dwdate, part, supplier WHERE p_category = 'MFGR#12' AND lo_orderdate = d_datekey AND lo_partkey = p_partkey AND lo_suppkey = s_suppkey AND s_region = 'AMERICA' GROUP BY d_year, p_brand ORDER BY d_year, p_brand;	2	3
SELECT SUM(lo_revenue), d_year, p_brand FROM lineorder, dwdate, part, supplier WHERE p_brand BETWEEN 'MFGR#2221' AND 'MFGR#2228' AND lo_orderdate = d_datekey AND lo_partkey = p_partkey AND lo_suppkey = s_suppkey AND s_region = 'ASIA' GROUP BY d_year, p_brand ORDER BY d_year, p_brand;	2	4
SELECT SUM(lo_revenue), d_year, p_brand FROM lineorder, dwdate, part, supplier WHERE p_brand = 'MFGR#2221' AND lo_orderdate = d_datekey AND lo_partkey = p_partkey AND lo_suppkey = s_suppkey AND s_region = 'EUROPE' GROUP BY d_year, p_brand ORDER BY d_year, p_brand;	2	5
SELECT c_nation, s_nation, d_year, SUM(lo_revenue) AS revenue FROM customer, lineorder, supplier, dwdate WHERE lo_custkey = c_custkey AND lo_suppkey = s_suppkey AND lo_orderdate = d_datekey AND c_region = 'ASIA' AND s_region = 'ASIA' AND d_year >= 1992 AND d_year <= 1997 GROUP BY c_nation, s_nation, d_year ORDER BY d_year asc, revenue DESC;	3	6

Continua...

Query	Grupo	ID
SELECT c_city, s_city, d_year, SUM(lo_revenue) AS revenue FROM customer, lineorder, supplier, dwdate WHERE c_nation = 'UNITED STATES' AND lo_custkey = c_custkey AND lo_suppkey = s_suppkey AND lo_orderdate = d_datekey AND s_nation = 'UNITED STATES' AND d_year >= 1992 AND d_year <= 1997 GROUP BY c_city, s_city, d_year ORDER BY d_year asc, revenue DESC;	3	7
SELECT c_city, s_city, d_year, SUM(lo_revenue) AS revenue FROM customer, lineorder, supplier, dwdate WHERE d_year >= 1992 AND lo_custkey = c_custkey AND lo_suppkey = s_suppkey AND lo_orderdate = d_datekey AND (c_city='UNITED KI1' OR c_city='UNITED KI5') AND (s_city='UNITED KI1' OR s_city='UNITED KI5') AND d_year <= 1997 GROUP BY c_city, s_city, d_year ORDER BY d_year asc, revenue DESC;	3	8
SELECT c_city, s_city, d_year, SUM(lo_revenue) AS revenue FROM customer, lineorder, supplier, dwdate WHERE d_yearmonth = 'Dec1997' AND lo_custkey = c_custkey AND lo_suppkey = s_suppkey AND lo_orderdate = d_datekey AND (c_city='UNITED KI1' OR c_city='UNITED KI5') AND (s_city='UNITED KI1' OR s_city='UNITED KI5') GROUP BY c_city, s_city, d_year ORDER BY d_year asc, revenue DESC;	3	9
SELECT d_year, c_nation, SUM(lo_revenue - lo_supplycost) AS profit FROM dwdate, customer, supplier, part, lineorder WHERE lo_custkey = c_custkey AND lo_suppkey = s_suppkey AND lo_partkey = p_partkey AND lo_orderdate = d_datekey AND c_region = 'AMERICA' AND s_region = 'AMERICA' AND (p_mfgr = 'MFGR#1' OR p_mfgr = 'MFGR#2') GROUP BY d_year, c_nation ORDER BY d_year, c_nation;	4	10
SELECT d_year, s_nation, p_category, SUM(lo_revenue - lo_supplycost) AS profit FROM dwdate, customer, supplier, part, lineorder WHERE lo_custkey = c_custkey AND lo_suppkey = s_suppkey AND lo_partkey = p_partkey AND lo_orderdate = d_datekey AND c_region = 'AMERICA' AND s_region = 'AMERICA' AND (d_year = 1997 OR d_year = 1998) AND (p_mfgr = 'MFGR#1' OR p_mfgr = 'MFGR#2') GROUP BY d_year, s_nation, p_category ORDER BY d_year, s_nation, p_category;	4	11

Continua...

Query	Grupo	ID
SELECT d_year, s_city, p_brand, SUM(lo_revenue - lo_supplycost) AS profit FROM dwdate, customer, supplier, part, lineorder WHERE lo_custkey = c_custkey AND lo_suppkey = s_suppkey AND lo_partkey = p_partkey AND lo_orderdate = d_datekey AND c_region = 'AMERICA' AND s_nation = 'UNITED STATES' AND (d_year = 1997 OR d_year = 1998) AND p_category = 'MFGR#14' GROUP BY d_year, s_city, p_brand ORDER BY d_year, s_city, p_brand;	4	12

Fim

3

METODOLOGIA

Neste capítulo, será abordado a metodologia utilizada para se realizar os experimentos. Na Seção 3.1, será mostrado uma comparação entre os SGBDs NewSQL disponíveis para utilização e critério para a escolha do mesmo. Na Seção 3.2, será discutido o SingleStore, o SGBD NewSQL utilizado nos experimentos. E na Seção 3.3, será mostrado qual foi o processo dos experimentos realizados.

3.1 ESCOLHA DO NEWSQL

Para alcançar o objetivo deste trabalho, primeiramente, foi realizada uma pesquisa dos SGBDs NewSQL na forma de DBaaS (Muhammed *et al.*, 2021). Com o intuito de investigar as suas características, compará-los e promover um *ranking* entre eles a fim de escolher o melhor candidato para o objeto de estudo.

Tal *ranking* foi obtido a partir do *website* DB-Engines (DB-Engines, 2022), o qual retornou 16 SGBDs NewSQL. Pela leitura da documentação dos 16 SGBDs (SingleStore, 2022; CloudSpanner, 2022; SAP, 2022; Labs, 2022; PingCAP, 2022; Data, 2022; MariaDB, 2022; NuoDB, 2022; Munchen, 2022; Umbra, 2022; HarperDB, 2022; Altibase, 2022; FairCom, 2022; CitusData, 2022), o SingleStore foi o SGBD que mais se adequou a proposta deste trabalho, pois este é um DBaaS e possui *ColumnStore* e *RowStore* como forma de armazenamento de dados. Além de possuir uma ampla variedade de *clusters* com diferentes configurações, fornecer a maior quantidade de créditos para utilizar o SGBD de forma gratuita e possuir suporte gratuito a todos os seus clientes.

3.2 SINGLESTORE

O SingleStore[1], antigamente conhecido como MemSQL[2], é um BD NewSQL, que possui duas versões, o SingleStoreDB Cloud e o SingleStoreDB Self-Managed. O SingleStoreDB

[1] https://www.singlestore.com/
[2] https://www.singlestore.com/blog/revolution/

Cloud é um DBaaS *Multi-cloud* que pode ser implantado na AWS[3], na Azure[4] ou no GCP[5]. Neste trabalho, o SingleStoreDB Cloud foi implantado somente na AWS na região US East 1 (N. Virginia). Por outro lado, o SingleStoreDB Self-Managed, é um BD manualmente implantado e gerenciado no seu próprio *hardware*. O SingleStore proporciona operações transacionais (OLTP), assim como, analíticas (OLAP), entregando escalabilidade, rapidez e flexibilidade dos sistemas NoSQL, como também, a linguagem SQL e propriedades ACID.

Sendo um BD SQL escalável e distribuído, provendo capacidades como, estruturas de dados *lock-free*, *code generation* e *Multi Version Concurrency Control*, o SingleStore foi categorizado como um sistema de Processamento Analítico Transacional Híbrido (Hybrid Transactional Analytical Processing (HTAP)[6]).

O SingleStore disponibiliza 33 tipos de *clusters* em seu serviço. Porém, somente 9 estão disponíveis no plano gratuito, são eles: S-00, S-0, S-1, S-2, S-4, S-6, S-8, S-12 e S-16. Seus *clusters* se diferenciam em suas configurações na quantidade de unidades de processamento (vCPUs) e na quantidade de memória RAM. As informações de *hardware* e custo dos *clusters* estão descritas na Tabela 4. Adicionalmente, o SingleStore cobra $0.023/mês por Gigabyte (GB) armazenado no Hard Drive (HD).

Tabela 4: *Clusters* do SingleStore

Cluster	vCPUs	RAM	Custo/Hora ($)
S-00	2	16 GBs	0.65
S-0	4	32 GBs	1.30
S-1	8	64 GBs	2.60
S-2	16	128 GBs	5.20
S-4	32	256 GBs	10.40
S-6	48	384 GBs	15.60
S-8	64	512 GBs	20.80
S-12	96	768 GBs	31.20
S-16	128	1 Terabyte	41.60

Em cada *cluster* há dois tipos de nós (*nodes*)[7], o *aggregator* e a *leaf*. Entre os *aggregators*, há o *Master Aggregator* e o *Child Aggregator*, o *Master* é responsável por distribuir as operações de cada consulta para as *leaf nodes*, agrupar os resultados enviados pelas mesmas e executar operações de Data Definition Language (DDL) e Data Manipulation Language (DML). Enquanto, o *Child Aggregator* pode executar operações de DML. Em compensação, as *leaf nodes* tem como

[3] https://aws.amazon.com/pt/
[4] https://azure.microsoft.com/pt-br/
[5] https://cloud.google.com/?hl=pt-br
[6] https://www.gartner.com/en/information-technology/glossary/htap-enabling-memory-computing-technologies
[7] https://www.singlestore.com/blog/understanding-memsql-in-5-easy-questions/

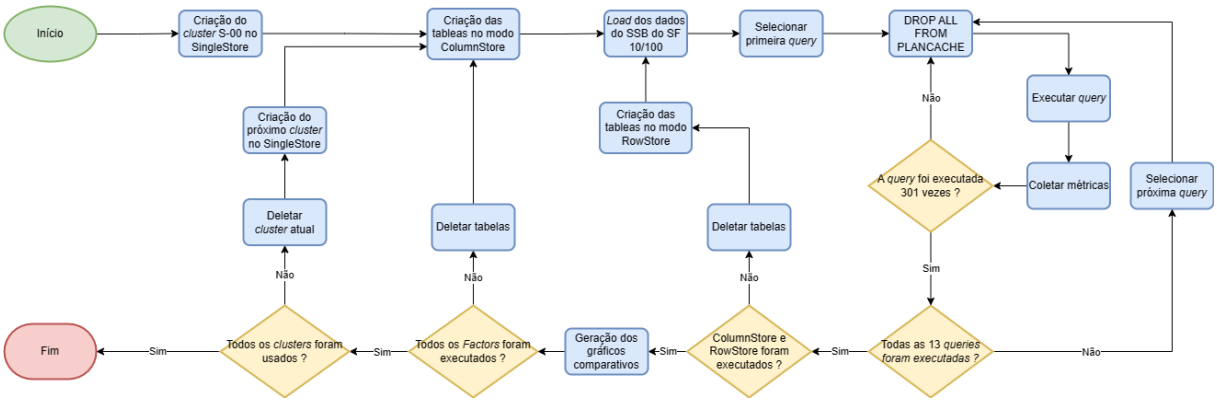
função armazenar os dados e computar as consultas recebidas.

3.3 EXPERIMENTOS

Para a realização dos experimentos foi definido um fluxo, apresentado na Figura 2, para que se fosse possível obter as métricas. O fluxo consiste em criar um *cluster* no SingleStoreDB Cloud na AWS na região US East 1 (N. Virginia), criar as tabelas no formato de armazenamento *ColumnStore* ou *RowStore*, inserir os dados nessas tabelas de acordo com o SF escolhido, executar cada consulta 301 vezes. Ressalta-se que, as métricas da primeira execução de cada consulta são descartadas, pois há um tempo maior necessário para processamento típico do SingleStore em virtude do seu *warmup*, a inicialização de seus *clusters*. Mas antes de cada execução, executar o comando "DROP ALL FROM PLANCACHE". Este comando tem como objetivo limpar todo o plano de execução armazenado na *cache* do SingleStore, para que em cada execução a consulta seja interpretada como se tivesse sido lida pelo SingleStore pela primeira vez. Finalizada a execução das 300 iterações, verifica-se se ambos os formatos de armazenamento foram executados, em caso negativo, deleta-se as tabelas, cria-as novamente no outro formato de armazenamento e coleta-se novamente as métricas para 300 iterações.

Coletadas as métricas dos dois formatos de armazenamento para o mesmo SF, geram-se os gráficos, segue-se para o próximo SF e reinicia-se o processo das tabelas e execuções. Concluído os dois SFs, prossegue-se para o próximo *cluster*, reiniciando o processo dos formatos de armazenamento, tabelas e execuções. E, terminado todos os *clusters*, finalizam-se os experimentos.

Figura 2: Fluxograma dos experimentos



O motivo para realizar as execuções 300 vezes, encontra-se respaldada no Teorema Central do Limite, que diz respeito à quantidade mínima de execuções de um experimento para se obter uma distribuição normal (Islam Rafiqul, 2018) da variável analisada.

As métricas coletadas foram: CPU_TIME_MS, ELAPSED_TIME_MS e AVG_MEMORY_USED_B, que correspondem ao tempo de execução na CPU (em milissegundos), tempo total decorrido de consulta (em milissegundos) e quantidade de memória RAM (em

Bytes), respectivamente. Para cada uma delas foi calculado o valor mínimo, máximo, desvio padrão, variância, covariância, mediana e média, assim como o intervalo mínimo e máximo da média para um intervalo de confiança de 95% das 300 execuções. De forma complementar, foi medido o volume de dados armazenados em disco e em memória RAM. Adicionalmente, gráficos de barra das médias com seus respectivos intervalos de confiança foram gerados para comparar os valores das métricas do *ColumnStore* e *RowStore* de cada SF.

Devido ao formato *RowStore* ser salvo na memória RAM e a quantidade variável de memória RAM disponibilizada em cada *cluster*, foi necessário o uso do *cluster* S-6. Pois este foi o primeiro *cluster* capaz de armazenar o SF100 para comparar as métricas entre *ColumnStore* e *RowStore*. Em contrapartida, para o SF10, foi possível comparar as métricas entre *ColumnStore* e *RowStore* para o SF10 dos *clusters* S-0, S-1, S-2, S-4 e S-6.

Os códigos para a realização dos experimentos, coleta das métricas, cálculo das medidas das métricas e a geração dos seus respectivos gráficos se encontram no Apêndice.

4

RESULTADOS

Neste capítulo, serão apresentados e discutidos os resultados dos experimentos descritos na Seção 3.3. Primeiramente, na Seção 4.1, será abordado o volume de dados necessário para armazenar os SFs em cada *cluster*. Em seguida, na Seção 4.2, será divulgado o tempo médio de execução na CPU de cada consulta. Na Seção 4.3, será apresentado a média do tempo total de consulta de cada consulta. Na Seção 4.4, será exposto a quantidade média de memória RAM consumida em cada consulta. Por fim, na Seção 4.5, serão discutidos todos os resultados obtidos nas seções anteriores. Em cada Seção são apresentados gráficos das métricas que são comparados pelo formato de armazenamento em cada *cluster* e em cada SF.

4.1 VOLUME DE DADOS

O SSB em sua forma original, possui 5.68GBs e 58GBs, nos SFs 10 e 100, respectivamente. Este volume de dados foi coletado quando o SSB foi inserido nos *clusters* do SingleStore, tais números são apresentados nas Tabelas 5 e 6.

Tabela 5: Volume dos dados do Star Schema Benchmark no Formato ColumnStore

		Index Disk Usage	Disk Usage
SF10	S-00	366 Megabytes (MBs)	3.5 GBs
	S-0	366 MBs	3.5 GBs
	S-1	352 MBs	3.5 GBs
	S-2	338 MBs	3.5 GBs
	S-4	326 MBs	3.5 GBs
	S-6	313 MBs	3.5 GBs
SF100	S-00	3.7 GBs	36 GBs
	S-0	3.7 GBs	36 GBs
	S-1	3.7 GBs	36.2 GBs
	S-2	3.6 GBs	36.3 GBs
	S-4	3.5 GBs	36.4 GBs
	S-6	3.4 GBs	36.3 GBs

Tabela 6: Volume dos dados do Star Schema Benchmark no Formato RowStore

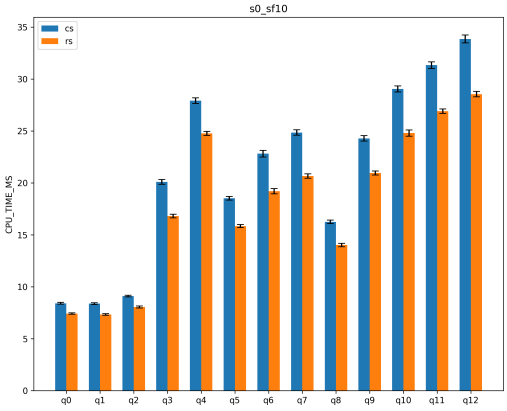
		Memory Usage
SF10	S-0	18.9 GBs
	S-1	18.9 GBs
	S-2	18.9 GBs
	S-4	18.9 GBs
	S-6	18.9 GBs
SF100	S-6	186.3 GBs

4.2 TEMPO DE CPU

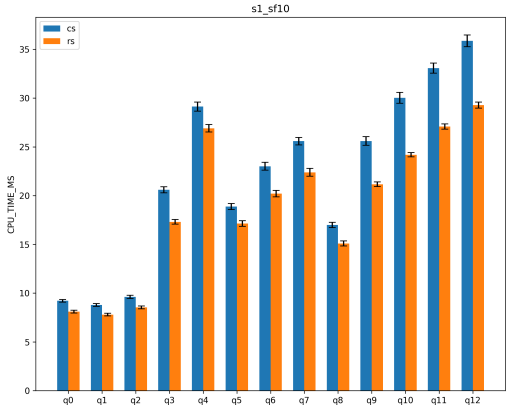
O tempo de CPU é importante para a avaliação do tempo necessário para o cálculo do resultado das consultas. As comparações estão mostradas nas Figuras 3 e 4. Os gráficos das Figuras a seguir têm no seu eixo *y*, a quantidade de tempo em milissegundos da execução da consulta na CPU, e no eixo *x*, a identificação das consultas. As barras azuis indicam o formato de armazenamento *ColumnStore* e as laranjas indicam o *RowStore*, as barras pretas no topo de cada coluna indicam o intervalo de confiança para cada coluna.

Figura 3: Tempo de CPU para o SF10

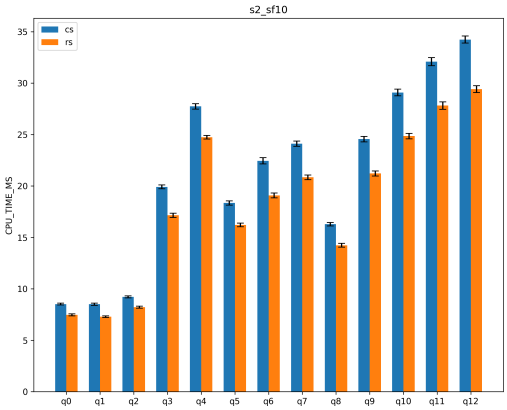
(a) S-0



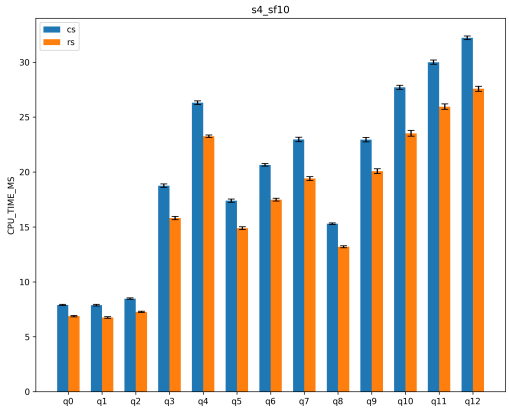
(b) S-1



(c) S-2



(d) S-4



(e) S-6

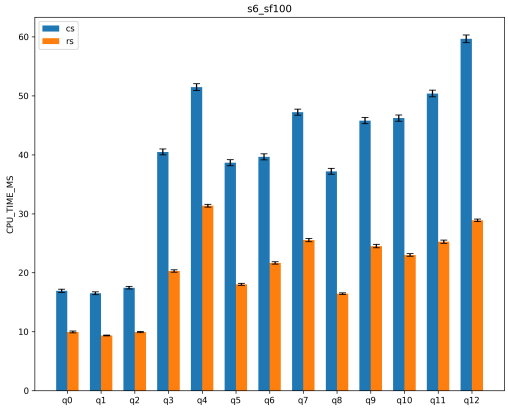
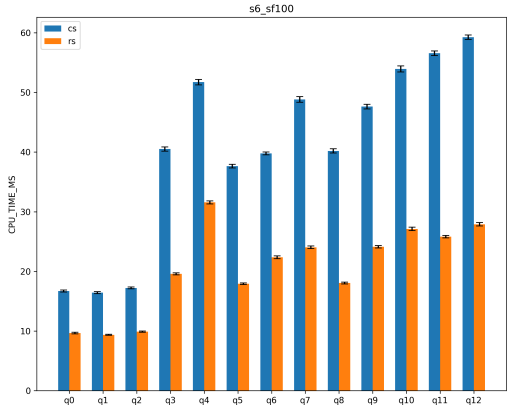


Figura 4: Tempo de CPU para o SF100

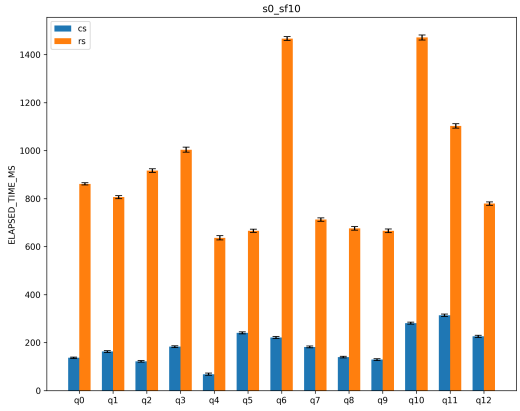


4.3 TEMPO TOTAL DE CONSULTA

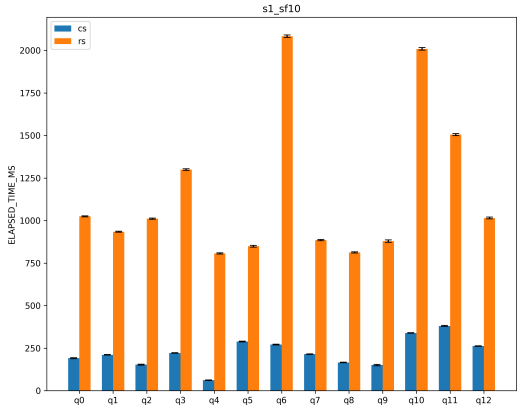
O tempo de execução na CPU, mostrado na Seção 4.2, compreende apenas uma parte do tempo total de consulta gasto pelas consultas no SingleStore. O tempo total de consulta, que engloba o tempo de operações de Entrada/Saída, tempo de espera e tempo de análises sintáticas e semânticas, pode ser visualizado nas Figuras 5 e 6. Os gráficos das Figuras a seguir têm no seu eixo *y*, a quantidade de tempo total de consulta em milissegundos, e no eixo *x*, a identificação das consultas. As barras azuis indicam o formato de armazenamento *ColumnStore* e as laranjas indicam o *RowStore*, as barras pretas no topo de cada coluna indicam o intervalo de confiança para cada coluna.

Figura 5: Tempo Total de Consulta do SF10

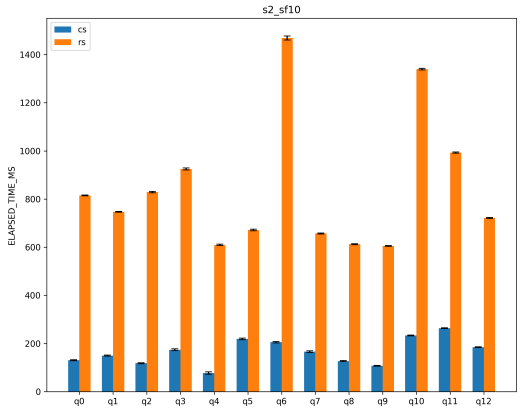
(a) S-0



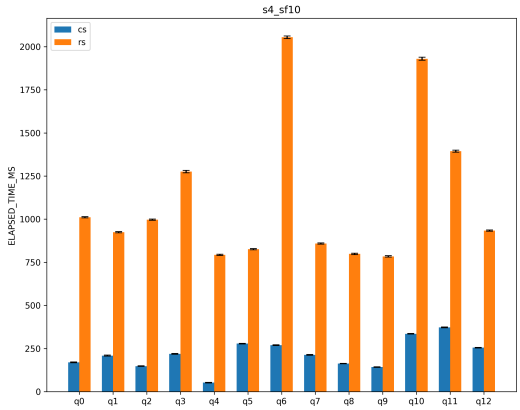
(b) S-1



(c) S-2



(d) S-4



(e) S-6

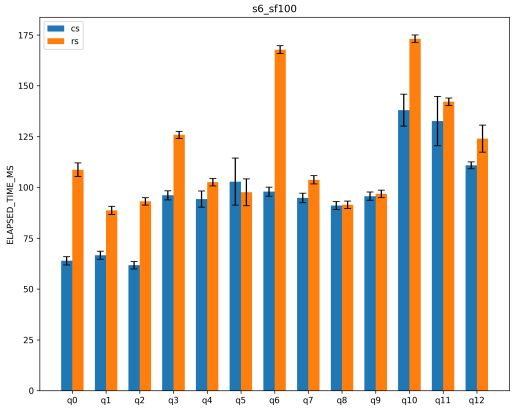
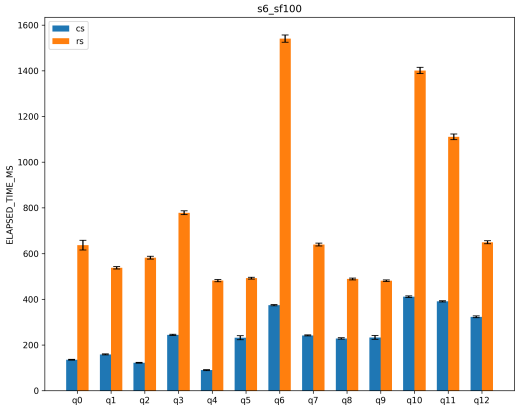


Figura 6: Tempo Total de Consulta do SF100

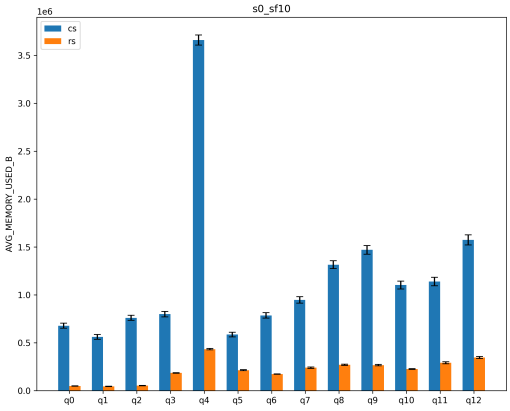


4.4 CONSUMO DE MEMÓRIA RAM

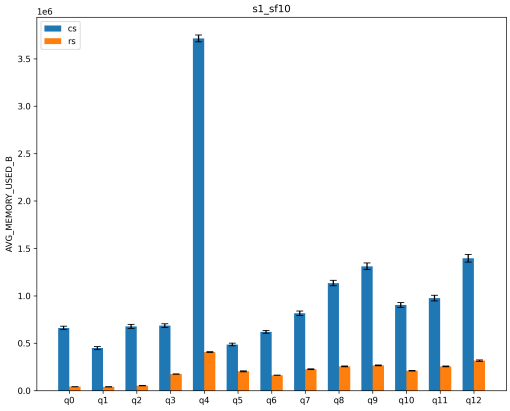
O consumo de memória RAM nas consultas, utilizada para armazenamento e posterior leitura dos dados, é mostrada nas Figuras 7 e 8. Os gráficos das Figuras a seguir têm no seu eixo *y*, a quantidade de Bytes utilizados na memória RAM, e no eixo *x*, a identificação das consultas. As barras azuis indicam o formato de armazenamento *ColumnStore* e as laranjas indicam o *RowStore*, as barras pretas no topo de cada coluna indicam o intervalo de confiança para cada coluna.

Figura 7: Consumo de Memória RAM do SF10

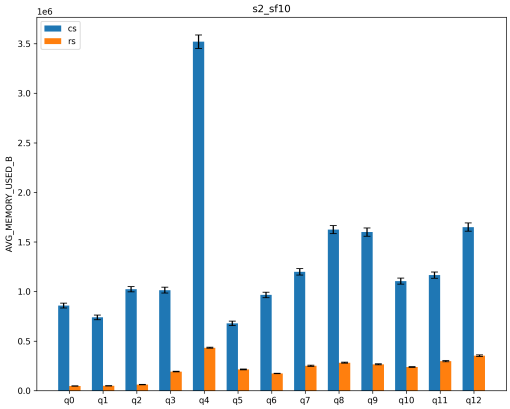
(a) S-0



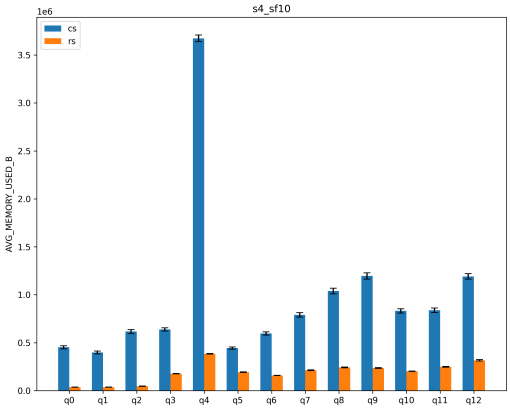
(b) S-1



(c) S-2



(d) S-4



(e) S-6

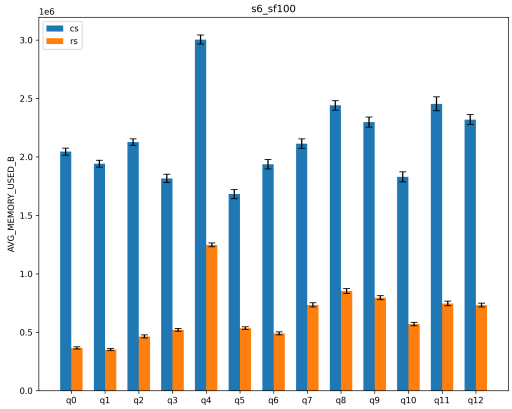
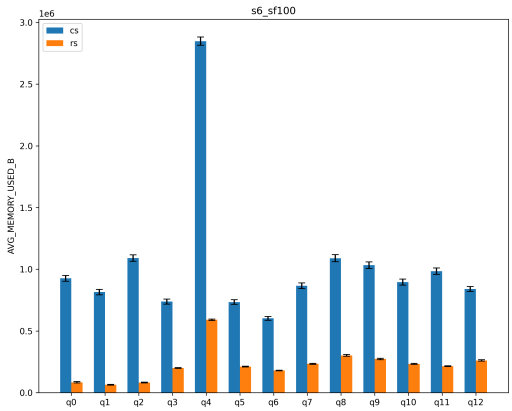


Figura 8: Consumo de Memória RAM do SF100



4.5 ANÁLISE

O volume de dados, para o *ColumnStore*, se mostrou menor que o volume original do SSB devido a sua compactação pelo método do SingleStore[1]. Para o SF 10, o SingleStore foi armazenou em média 3.5GBs de dados para o SF 10, uma redução de 40%. Para o SF 100, necessitou armazenar em média 36.2GBs de dados, uma redução de 38%. Pode-se ver pela Tabela 5, que o *Index Disk Usage* variou de tamanho nos *clusters*, isto teve motivo aos comandos utilizados para criar as tabelas, que não possuíam a especificação de qual(is) coluna(s) usar como índice, portanto os índices foram gerados automaticamente pelo SingleStore. Consequentemente, isto pode ter influenciado no valor das outras métricas.

Para o formato RowStore, o SingleStore necessitou de mais armazenamento do que o volume original dos dados do SSB. Isto se deve a equação[2] usada para calcular tal valor do volume de dados por tabela: $(data + 64\ bytes + (index\ size * number\ of\ secondary\ indexes)) * number\ of\ rows$. Onde na fórmula, temos que: *data* é o tamanho dos dados com base no número de colunas e seus respectivos tipos de dados; 64 bytes são em relação aos metadados e índices das chaves primárias; *index size* é o tamanho do tipo de índice de quaisquer índices adicionais na tabela; *number of secondary indexes* é o número de índices adicionais; e, *number of rows* é o número total de linhas em uma determinada tabela. O custo de armazenagem para o formato *ColumnStore* da Tabela 5 é mostrado na Tabela 7.

A partir dos gráficos gerados pelos resultados dos experimentos, pode-se concluir os fatos a seguir baseados por observação.

Para a métrica CPU_TIME_MS, é possível identificar que o formato *RowStore* necessita de menos tempo de execução na CPU do que o formato *ColumnStore*. A diferença é pouca entre

[1] https://docs.singlestore.com/db/v8.0/en/create-a-database/columnstore/columnstore-sizing-estimations.html

[2] https://docs.singlestore.com/managed-service/en/create-a-database/rowstore.html

Tabela 7: Valor do custo de armazenagem do Star Schema Benchmark

VOLUME (GBs)	CUSTO/MÊS ($)
3.5	0.0805
36	0.828
36.2	0.8326
36.3	0.8349

os dois formatos com exceção para o *cluster* S-6, onde em algumas consultas a redução de tempo atingiu mais de 50%.

Para a métrica ELAPSED_TIME_MS, houve uma inversão na ordem, dessa vez, é notado que o *RowStore* possui maior tempo de execução total de consulta para todas as consultas em todos os SFs, com exceção para a consulta de ID 5 no SF 10 para o *cluster* S-6. Para o caso da consulta de ID 5, pode ter acontecido uma nova atribuição da consulta para outra *leaf node* em pelo menos uma das 300 execuções da consulta, o que causaria no aumento do seu tempo total de consulta.

Por último, para a métrica AVG_MEMORY_USED_B, o *ColumnStore* sempre exibiu um valor maior do que o *RowStore*, com atenção especial a consulta de ID 4, que em todos os casos apresentou um valor bem maior do que o *RowStore*.

5

CONCLUSÃO

Neste trabalho, o foco de estudo foi a avaliação do desempenho dos *clusters* do SGBD NewSQL SingleStoreDB Cloud para o *benchmark* SSB entre os formatos de armazenamento *RowStore* e *ColumnStore*. Para isto, as 13 consultas do SSB foram executadas 300 vezes para cada formato de armazenamento em 5 *clusters* do SingleStore e nos SFs 10 e 100, onde foi medido o tempo de CPU, tempo total de consulta e quantidade de memória RAM usada. Com relação aos *clusters*, foi realizado o escalonamento vertical dos 5 *clusters* para verificar a correspondência entre o aumento do número de CPUs e tamanho da memória RAM com a melhora das métricas das consultas.

Analisando-se somente os formatos de armazenamento, os resultados mostraram que os tempos de CPU entre os dois formatos se apresentaram bastante semelhantes entre si. De tal modo que na maioria dos casos, o *RowStore* foi ligeiramente mais rápido em suas execuções na CPU do que o *ColumnStore*. Para os tempos totais de consulta, o *RowStore* contabilizou mais tempo necessário do que o *ColumnStore*. O consumo de RAM foi bem maior no *ColumnStore* do que no *RowStore*, já que os dados precisam ser transportados da memória secundária, HD, para a memória principal, RAM, o que ocasionou na intensificação do uso da RAM. Estes resultados mostram que o *RowStore* se saiu vantajoso em 2 das 3 métricas avaliadas em comparação ao *ColumnStore* para todos os *clusters*.

Quando comparado os gráficos das métricas para cada *cluster*, nenhum dos *clusters* se sobressaiu no tempo de execução na CPU, tendo todos praticamente os mesmos tempos. Para o tempo total de consulta, o *cluster* S-6 apresentou tempos menores no formato *ColumnStore* do que os outros *clusters*, porém, não se viu uma diminuição no tempo conforme progrediu-se o poder de processamento e memória RAM dos *clusters*. O consumo de memória RAM foi menor no *cluster* S-4 no formato *RowStore*, e novamente não foi possível ver uma correspondência na diminuição do consumo ao aumentar a capacidade do *cluster*.

Uma observação adicional reside no custo do uso do SingleStore, que cobra por hora proporcionalmente a configuração do *cluster* e por GB armazenado em HD. Isso faz com que, se um usuário deseja o emprego do formato *RowStore*, quanto maior for o volume dos dados, maior serão os custos de operação do *cluster* deste usuário. Logo, há um *trade-off* entre manter os dados no formato *RowStore* e pagar mais caro pela configuração do *cluster*, e manter os dados

no formato *ColumnStore* e pagar mais barato no *cluster*, mas, pagar pelo seu armazenamento, o que pode se equiparar ou se sobressair ao valor do custo do *RowStore* dependendo da quantidade de dados armazenados no HD. Vale lembrar que, não somente o tipo de armazenamento pode ser fator determinante para a escolha do *cluster*, mas também, como a necessidade de mais processamento (*vCPUs*) para realizar as operações sobre os dados.

Como trabalho futuro, pretende-se avaliar as mesmas métricas deste estudo para outros *clusters* de SGBDs DBaaS NewSQL. Com o propósito de gerar uma comparação entre os diferentes tipos de *clusters* e escalonamento vertical e horizontal para promover uma análise baseado tanto no desempenho sobre as métricas quanto nos custos de seu funcionamento.

REFERÊNCIAS

Abadi, D. J., Madden, S. R., & Hachem, N. (2008). Column-Stores vs. Row-Stores: How Different Are They Really? *Proceedings of the ACM SIGMOD Conference on Management of Data*, 967–980.

Altibase (2022). *Altibase*. `https://docs.altibase.com/display/arch/Home` [Acesso em: 3 jul. 2022.].

Binani, S., Gutti, A., & Upadhyay, S. (2016). SQL vs. NoSQL vs. NewSQL - A Comparative Study. *Communications on Applied Electronics*, 6(1):43–46.

CitusData (2022). *Citus*. `https://docs.citusdata.com/en/stable/` [Acesso em: 5 jul. 2022.].

CloudSpanner (2022). *Google*. `https://cloud.google.com/spanner/docs?hl=pt-br` [Acesso em: 24 jun. 2022.].

Codd, E. F. (1970). A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13(6):377–387.

Data, V. A. (2022). *VoltDB*. `https://docs.voltdb.com/` [Acesso em: 26 jun. 2022.].

DB-Engines (2022). *DB-Engines Ranking*. `https://db-engines.com/en/ranking` [Acesso em: 23 jun. 2022.].

FairCom (2022). *FairCom DB*. `https://learn.faircomcorp.com/developers/documentation_directory` [Acesso em: 4 jul. 2022.].

Han, J., E, H., Le, G., & Du, J. (2011). Survey on NoSQL database. *Conference on Pervasive Computing and Applications*, 6:363–366.

HarperDB (2022). *HarperDB*. `https://docs.harperdb.io/docs/` [Acesso em: 1 jul. 2022.].

Islam Rafiqul, M. (2018). Sample Sizes and Its Role in Central Limit Theorem (CLT). *Computional and Applied Mathematics Journal*, 4(1):1–7.

Kaur, K. & Sachdeva, M. (2017). Performance Evaluation of NewSQL Databases. *International Conference on Inventive Systems and Control*, 1–5.

Kim, W. (2009). Cloud Computing: Today and Tomorrow. *Journal Of Object Technology*, 8(1):65–72.

Kimball, R. & Ross, M. (2013). *The Data Warehouse Toolkit*. Wiley, 3 edition.

Labs, C. (2022). *Cockroach*. `https://www.cockroachlabs.com/docs/stable/` [Acesso em: 23 jun. 2022.].

MariaDB (2022). *SkySQL*. `https://mariadb.com/docs/skysql/` [Acesso em: 30 jun. 2022.].

Moniruzzaman, A. B. M. (2014). NewSQL: Towards Next-Generation Scalable RDBMS for Online Transaction Processing (OLTP) for Big Data Management. *International Journal of Database Theory & Application*, 7(6):121–130.

Muhammed, A., Hilmi Abdullah, Z., Ismail, W., Y. Aldailamy, A., Radman, A., Hendradi, R., & Rafiee Afandi, R. (2021). A Survey of NewSQL DBMSs focusing on Taxonomy, Comparison and Open Issues. *Journal of Computer Science & Computational Mathematics*, 11(4).

Munchen, T. U. (2022). *Hyper*. `https://hyper-db.de/index.html` [Acesso em: 1 jul. 2022.].

NuoDB (2022). *NuoDB*. `https://doc.nuodb.com/nuodb/latest/introduction-to-nuodb/` [Acesso em: 28 jun. 2022.].

Pavlo, A. & Aslett, M. (2016). What's Really New With NewSQL? *ACM Sigmod Record*, 45(2):45–55.

PingCAP (2022). *TiDB*. `https://docs.pingcap.com/tidb/dev` [Acesso em: 25 jun. 2022.].

SAP (2022). *SAP HANA*. `https://help.sap.com/docs/HANA_SERVICE_CF?locale=en-US` [Acesso em: 24 jun. 2022.].

SingleStore (2022). *SingleStoreDB Cloud*. `https://docs.singlestore.com/managed-service/en/getting-started-with-singlestoredb-cloud/about-singlestoredb-cloud.html` [Acesso em: 23 jun. 2022.].

S.Kanade, M. A. & Gopal, D. A. (2013). Chossing Right Database System: Row or Column-Store. *Proceedings of the International Conference on Information Communication and Embedded Systems*, 16–20.

Sterjo, K. (2017). Row-Store / Column-Store / Hybrid-Store.

Stonebraker, M., Madden, S., Abadi, D. J., Harizopoulos, S., & Helland, P. (2007). The End of an Architectural Era (It's Time for a Complete Rewrite. *Proceedings of the 33rd International Conference on Very Large Data Bases*, 33:1150–1160.

Strauch, C. (2011). *NoSQL Databases*. `http://www.christof-strauch.de/nosqldbs.pdf` [Acesso em: 8 ago. 2022.].

Umbra (2022). *Umbra*. `https://umbra-db.com/` [Acesso em: 1 jul. 2022.].

Apêndice

A

CÓDIGO DAS MÉTRICAS

```
import mysql.connector
import pandas as pd
import numpy as np
from scipy.stats import t, variation
from datetime import datetime
import os
import credentials

def ci(data, cf=0.95):
   m = np.mean(data)
   s = np.std(data)
   l = len(data)
   t_crit = np.abs(t.ppf((1-cf)/2,l-1))
   return m, m-s*t_crit/np.sqrt(l), m+s*t_crit/np.sqrt(l)

queries = [
   "select sum(lo_extendedprice*lo_discount) as revenue from
      ↪ lineorder, dwdate where d_yearmonthnum = 199401 and
      ↪ lo_orderdate = d_datekey and lo_discount between 4 and
      ↪  6 and lo_quantity between 26 and 35;",
   "select sum(lo_extendedprice*lo_discount) as revenue from
      ↪ lineorder, dwdate where d_year = 1993 and lo_orderdate
      ↪  = d_datekey and lo_discount between 1 and 3 and
      ↪ lo_quantity < 25;",
   "select sum(lo_extendedprice*lo_discount) as revenue from
      ↪ lineorder, dwdate where d_weeknuminyear = 6 and
      ↪ lo_orderdate = d_datekey and d_year = 1994 and
      ↪ lo_discount between 5 and 7 and lo_quantity between 26
      ↪  and 35;",
```

```
"select sum(lo_revenue), d_year, p_brand from lineorder,
   ↪ dwdate, part, supplier where p_category = 'MFGR#12'
   ↪ and lo_orderdate = d_datekey and lo_partkey =
   ↪ p_partkey and lo_suppkey = s_suppkey and s_region = '
   ↪ AMERICA' group by d_year, p_brand order by d_year,
   ↪ p_brand;",
"select sum(lo_revenue), d_year, p_brand from lineorder,
   ↪ dwdate, part, supplier where p_brand between 'MFGR
   ↪ #2221' and 'MFGR#2228' and lo_orderdate = d_datekey
   ↪ and lo_partkey = p_partkey and lo_suppkey = s_suppkey
   ↪ and s_region = 'ASIA' group by d_year, p_brand order
   ↪ by d_year, p_brand;",
"select sum(lo_revenue), d_year, p_brand from lineorder,
   ↪ dwdate, part, supplier where p_brand = 'MFGR#2221' and
   ↪  lo_orderdate = d_datekey and lo_partkey = p_partkey
   ↪ and lo_suppkey = s_suppkey and s_region = 'EUROPE'
   ↪ group by d_year, p_brand order by d_year, p_brand;",
"select c_nation, s_nation, d_year, sum(lo_revenue) as
   ↪ revenue from customer, lineorder, supplier, dwdate
   ↪ where lo_custkey = c_custkey and lo_suppkey =
   ↪ s_suppkey and lo_orderdate = d_datekey and c_region =
   ↪ 'ASIA' and s_region = 'ASIA' and d_year >= 1992 and
   ↪ d_year <= 1997 group by c_nation, s_nation, d_year
   ↪ order by d_year asc, revenue desc;",
"select c_city, s_city, d_year, sum(lo_revenue) as revenue
   ↪ from customer, lineorder, supplier, dwdate where
   ↪ c_nation = 'UNITED STATES' and lo_custkey = c_custkey
   ↪ and lo_suppkey = s_suppkey and lo_orderdate =
   ↪ d_datekey and s_nation = 'UNITED STATES' and d_year >=
   ↪  1992 and d_year <= 1997 group by c_city, s_city,
   ↪ d_year order by d_year asc, revenue desc;",
"select c_city, s_city, d_year, sum(lo_revenue) as revenue
   ↪ from customer, lineorder, supplier, dwdate where
   ↪ d_year >= 1992 and lo_custkey = c_custkey and
   ↪ lo_suppkey = s_suppkey and lo_orderdate = d_datekey
   ↪ and (c_city='UNITED KI1' or c_city='UNITED KI5') and (
   ↪ s_city='UNITED KI1' or s_city='UNITED KI5') and d_year
   ↪  <= 1997 group by c_city, s_city, d_year order by
   ↪ d_year asc, revenue desc;",
```

```
    "select c_city, s_city, d_year, sum(lo_revenue) as revenue
       ↪ from customer, lineorder, supplier, dwdate where
       ↪ d_yearmonth = 'Dec1997' and lo_custkey = c_custkey and
       ↪  lo_suppkey = s_suppkey and lo_orderdate = d_datekey
       ↪ and (c_city='UNITED KI1' or c_city='UNITED KI5') and (
       ↪ s_city='UNITED KI1' or s_city='UNITED KI5') group by
       ↪ c_city, s_city, d_year order by d_year asc, revenue
       ↪ desc;",
    "select d_year, c_nation, sum(lo_revenue - lo_supplycost) as
       ↪  profit from dwdate, customer, supplier, part,
       ↪ lineorder where lo_custkey = c_custkey and lo_suppkey
       ↪ = s_suppkey and lo_partkey = p_partkey and
       ↪ lo_orderdate = d_datekey and c_region = 'AMERICA' and
       ↪ s_region = 'AMERICA' and (p_mfgr = 'MFGR#1' or p_mfgr
       ↪ = 'MFGR#2') group by d_year, c_nation order by d_year,
       ↪  c_nation;",
    "select d_year, s_nation, p_category, sum(lo_revenue -
       ↪ lo_supplycost) as profit from dwdate, customer,
       ↪ supplier, part, lineorder where lo_custkey = c_custkey
       ↪  and lo_suppkey = s_suppkey and lo_partkey = p_partkey
       ↪  and lo_orderdate = d_datekey and c_region = 'AMERICA'
       ↪  and s_region = 'AMERICA' and (d_year = 1997 or d_year
       ↪  = 1998) and (p_mfgr = 'MFGR#1' or p_mfgr = 'MFGR#2')
       ↪ group by d_year, s_nation, p_category order by d_year,
       ↪  s_nation, p_category;",
    "select d_year, s_city, p_brand, sum(lo_revenue -
       ↪ lo_supplycost) as profit from dwdate, customer,
       ↪ supplier, part, lineorder where lo_custkey = c_custkey
       ↪  and lo_suppkey = s_suppkey and lo_partkey = p_partkey
       ↪  and lo_orderdate = d_datekey and c_region = 'AMERICA'
       ↪  and s_nation = 'UNITED STATES' and (d_year = 1997 or
       ↪ d_year = 1998) and p_category = 'MFGR#14' group by
       ↪ d_year, s_city, p_brand order by d_year, s_city,
       ↪ p_brand;"
    ]

base_query = "SELECT CPU_TIME_MS, CPU_WAIT_TIME_MS,
    ↪ ELAPSED_TIME_MS, DISK_LOGICAL_READ_B,
    ↪ DISK_LOGICAL_WRITE_B, DISK_PHYSICAL_READ_B,
```

```
   ↪ DISK_PHYSICAL_WRITE_B, 1000*MEMORY_BS/ELAPSED_TIME_MS
   ↪ AVG_MEMORY_USED_B FROM INFORMATION_SCHEMA.
   ↪ mv_query_activities_extended_cumulative WHERE QUERY_TEXT
   ↪ LIKE "
queries_text = [
   "\"select sum(lo_extendedprice*lo_discount) as revenue from
      ↪ lineorder, dwdate where d_yearmonthnum%\"",
   "\"select sum(lo_extendedprice*lo_discount) as revenue from
      ↪ lineorder, dwdate where d_year%\"",
   "\"select sum(lo_extendedprice*lo_discount) as revenue from
      ↪ lineorder, dwdate where d_weeknuminyear%\"",
   "\"select sum(lo_revenue), d_year, p_brand from lineorder,
      ↪ dwdate, part, supplier where p_category%\"",
   "\"select sum(lo_revenue), d_year, p_brand from lineorder,
      ↪ dwdate, part, supplier where p_brand between%\"",
   "\"select sum(lo_revenue), d_year, p_brand from lineorder,
      ↪ dwdate, part, supplier where p_brand =%\"",
   "\"select c_nation%\"",
   "\"select c_city, s_city, d_year, sum(lo_revenue) as revenue
      ↪  from customer, lineorder, supplier, dwdate where
      ↪ c_nation%\"",
   "\"select c_city, s_city, d_year, sum(lo_revenue) as revenue
      ↪  from customer, lineorder, supplier, dwdate where
      ↪ d_year%\"",
   "\"select c_city, s_city, d_year, sum(lo_revenue) as revenue
      ↪  from customer, lineorder, supplier, dwdate where
      ↪ d_yearmonth%\"",
   "\"select d_year, c_nation%\"",
   "\"select d_year, s_nation%\"",
   "\"select d_year, s_city%\""
]

host = credentials.HOST
user = credentials.USER
password = credentials.PASSWORD
w_type = "s6"
database = "sf10cs"
n_query = 300
```

```
conn = mysql.connector.connect(host=host, user=user, password=
    ↪ password, database=database)
if conn.is_connected():
    try:
        print('Connected')
        for i in range(0,13):
            exec = []
            start = datetime.today()
            print("start_{}:{}".format(i, start.strftime("%Y-%m-%d
                ↪  %H:%M:%S")))
            for _ in range(0, n_query+1):
                go = True
                while go:
                    cursor = conn.cursor(buffered=True)
                    cursor.execute('DROP ALL FROM PLANCACHE')
                    cursor.execute(queries[i])
                    cursor.execute(base_query + queries_text[i])
                    res_fetchall = cursor.fetchall()
                    try:
                        res = {"CPU_TIME_MS": int(res_fetchall[0][0])
                            ↪ , "ELAPSED_TIME_MS": int(res_fetchall
                            ↪ [0][2]), "AVG_MEMORY_USED_B":
                            ↪ res_fetchall[0][7]}
                        pd_res = pd.DataFrame.from_records([res],
                            ↪ coerce_float=True)
                        exec.append(pd_res)
                    except:
                        pass
                    else:
                        go = False
                    finally:
                        cursor.close()
            results = pd.concat(exec)
            results.to_csv("data/{}/{}/{}/{}.csv".format(w_type,
                ↪ n_query,database,i), index=False, header=True)
            results = results.iloc[1: , :] # eliminando a primeira
                ↪  linha (valor que eh sempre muito alto) para
                ↪ calcular as metricas
```

```
            analysis = {"mean": [], "mean_min": [], "mean_max":
               ↪ [], "std": [], "var": [], "cv": [], "min": [], "
               ↪ max": [], "median": []}
            atts = ["CPU_TIME_MS", "ELAPSED_TIME_MS", "
               ↪ AVG_MEMORY_USED_B"]
            for att in atts:
               res = ci(results[att])
               analysis["mean"].append(res[0])
               analysis["mean_min"].append(res[1])
               analysis["mean_max"].append(res[2])
               analysis["std"].append(np.std(results[att]))
               analysis["var"].append(np.var(results[att]))
               analysis["cv"].append(variation(results[att]))
               analysis["min"].append(np.min(results[att]))
               analysis["max"].append(np.max(results[att]))
               analysis["median"].append(np.median(results[att]))
            analysis = pd.DataFrame.from_dict(analysis, orient='
               ↪ index', columns=atts)
            analysis.to_csv("data/{}/{}/{}/analysis_{}.csv".format
               ↪ (w_type,n_query,database,i), index=True, header=
               ↪ True)
            end = datetime.today()
            print("end_{}:{}".format(i, end.strftime("%Y-%m-%d %H
               ↪ :%M:%S")))
            delta = str(end - start).split(".")[0]
            times = {"START": start.strftime("%Y-%m-%d %H:%M:%S"),
               ↪  "FINISH": end.strftime("%Y-%m-%d %H:%M:%S"), "
               ↪ DELTA": delta}
            pd.DataFrame.from_dict([times]).to_csv("data/{}/{}/{}/
               ↪ times.csv".format(w_type,n_query,database), mode
               ↪ ="a", index=False, header=not os.path.exists(f"C
               ↪ :/Users/nicho/Documents/tg_newsql/data/{w_type}/{
               ↪ n_query}/{database}/times.csv"))
      except Exception as e:
         print(repr(e))
      finally:
         conn.close()
else:
      print("Error: not connected")
```

B

CÓDIGO DOS GRÁFICOS

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

plt.rcParams['errorbar.capsize'] = 4
queries = ['q'+str(a) for a in range(0,13)]
for c in ['s0', 's1', 's2', 's4', 's6']:
   for i, val in enumerate(['CPU_TIME_MS','ELAPSED_TIME_MS','
      ↪ AVG_MEMORY_USED_B']):
      h1, h2, e1, e2 = [], [], [], []
      for n in [str(a) for a in range(0,13)]:
         df1 = pd.read_csv(f"data/{c}/300/sf10cs/analysis_{n}.
            ↪ csv", index_col=0)
         df2 = pd.read_csv(f"data/{c}/300/sf10rs/analysis_{n}.
            ↪ csv", index_col=0)
         h1.append(df1.iloc[0][i])
         h2.append(df2.iloc[0][i])
         e1.append(df1.iloc[0][i] - df1.iloc[1][i])
         e2.append(df2.iloc[0][i] - df2.iloc[1][i])

      offsets = (np.arange(2)-np.arange(2).mean())/(2+1.)
      width = np.diff(offsets).mean()
      x = np.arange(13)
      fig, ax = plt.subplots(figsize=(10, 8))
      plt.bar(x + offsets[0], h1, width = width, yerr=e1, label
         ↪ ='cs')
      plt.bar(x + offsets[1], h2, width = width, yerr=e2, label
         ↪ ='rs')
      plt.ylabel(val)
```

```
        plt.xticks(x, queries)
        plt.legend(loc='upper left')
        plt.title(f"{c}_sf10")
        plt.savefig(f"data/{c}/300/sf10_{val}_mconf.png", dpi
            ↪ =300)
        plt.close()

for i, val in enumerate(['CPU_TIME_MS','ELAPSED_TIME_MS','
    ↪ AVG_MEMORY_USED_B']):
        for f in ['10', '100']:
            h1, h2, e1, e2 = [], [], [], []
            for n in [str(a) for a in range(0,13)]:
                df1 = pd.read_csv(f"data/s6/300/sf{f}cs/analysis_{n
                    ↪ }.csv", index_col=0)
                df2 = pd.read_csv(f"data/s6/300/sf{f}rs/analysis_{n
                    ↪ }.csv", index_col=0)
                h1.append(df1.iloc[0][i])
                h2.append(df2.iloc[0][i])
                e1.append(df1.iloc[0][i] - df1.iloc[1][i])
                e2.append(df2.iloc[0][i] - df2.iloc[1][i])
            offsets = (np.arange(2)-np.arange(2).mean())/(2+1.)
            width = np.diff(offsets).mean()
            x = np.arange(13)
            fig, ax = plt.subplots(figsize=(10, 8))
            plt.bar(x + offsets[0], h1, width = width, yerr=e1,
                ↪ label='cs')
            plt.bar(x + offsets[1], h2, width = width, yerr=e2,
                ↪ label='rs')
            plt.ylabel(val)
            plt.xticks(x, queries)
            plt.legend(loc='upper left')
            plt.title(f"s6_sf100")
            plt.savefig(f"data/s6/300/sf{f}_{val}_mconf.png", dpi
                ↪ =300)
            plt.close()
```