UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CAMPUS DOIS VIZINHOS
CURSO DE ESPECIALIZAÇÃO EM CIÊNCIA DE DADOS

DIEGO ALONSO

UMA AVALIAÇÃO EXPERIMENTAL DE EXECUÇÃO DE CONSULTAS OLAP NO POSTGRESQL E MONGODB

TRABALHO DE CONCLUSÃO DE CURSO DE ESPECIALIZAÇÃO

DOIS VIZINHOS
2022

DIEGO ALONSO

UMA AVALIAÇÃO EXPERIMENTAL DE EXECUÇÃO DE CONSULTAS OLAP NO POSTGRESQL E MONGODB

AN EXPERIMENTAL EVALUATION OF OLAP QUERY EXECUTION IN POSTGRESQL AND MONGODB

Trabalho de Conclusão de Curso de Especialização apresentado ao Curso de Especialização em Ciência de Dados da Universidade Tecnológica Federal do Paraná, como requisito para a obtenção do título de Especialista em Ciência de Dados.

Orientador: Prof. Dr. Yuri Kaszubowski Lopes

DOIS VIZINHOS
2022

DIEGO ALONSO

UMA AVALIAÇÃO EXPERIMENTAL DE EXECUÇÃO DE CONSULTAS OLAP NO POSTGRESQL E MONGODB

Trabalho de Conclusão de Curso de Especialização apresentado ao Curso de Especialização em Ciência de Dados da Universidade Tecnológica Federal do Paraná, como requisito para a obtenção do título de Especialista em Ciência de Dados.

Data de aprovação: 09/novembro/2022

Yuri Kaszubowski Lopes
Doutorado
Universidade do Estado de Santa Catarina

Rafael Alves Paes Oliveira
Doutorado
Universidade Tecnológica Federal do Paraná - Câmpus Dois Vizinhos

Rodolfo Adamshuk Silva
Doutorado
Universidade Tecnológica Federal do Paraná - Câmpus Dois Vizinhos

DOIS VIZINHOS
2022

AGRADECIMENTOS

Em primeiro lugar, agradeço a Deus pela pelo dom da vida e por ter me proporcionado chegar até aqui.

Em segundo lugar, gostaria de agradecer à minha noiva e companheira para a vida toda, Kriscia Maria Prestes Campos, por me acompanhar em todos os momentos da minha vida nestes últimos 10 anos, sendo eles momentos de alegrias, tristezas, dores e entre outros. Sempre me auxiliando de diversas maneiras, principalmente me encorajando e motivando para desenvolver este trabalho de conclusão.

Em terceiro, gostaria de agradecer aos meus pais e minha irmã, Fernando Henrique Alonso, Luciene Regina Leineker e Gabriele Alonso, que, desde sempre, me acompanham nos bons e turbulentos momentos de minha vida e apoiam em todas as minhas decisões.

A Universidade Tecnológica Federal do Paraná, por ter me dado a oportunidade de entrar no curso de Especialização em Ciência de Dados e por ter oferecido sempre uma ótima qualidade de ensino, com professores renomados e uma plataforma de ensino-aprendizagem bem estruturada para a educação a distância.

Por fim, a todos os meus professores e colegas que fizeram parte do período em que estive estudando na instituição. Em especial ao professor e doutor Anderson Chaves Carniel por ter me apoiado durante toda esta jornada do curso, principalmente na realização deste trabalho de conclusão, me mostrando sempre a intuição científica e os conceitos-chave necessários para a implementação do meu estudo e também ao professor doutor Yuri Kaszubowski Lopes por assumir na reta final de entrega deste trabalho dado a saída do professor Anderson.

RESUMO

No decorrer das últimas décadas o gerenciamento de dados vem se tornando uma das atividades mais importantes nas organizações, devido ao crescimento na geração de dados pelo mundo. Atualmente, o maior desafio existente é de organizar as informações e manipula-las com objetivo de auxiliar na tomada de decisões. Neste cenário, o *Data Warehouse* (DW) é um importante componente para apoiar as decisões. O DW é uma base de dados não volátil, histórica e massiva, orientada para o assunto, cujo processamento de consultas analíticas resulta em tempos de resposta elevados. Técnicas são utilizadas para melhorar o desempenho de processamento de consultas, entre elas estão o uso de fragmentação de dados, visões materializadas e índices. Além disso, o NoSQL é uma tecnologia emergente cujas principais características são o aprimoramento do processamento de consultas e armazenamento de dados, tornando-se uma alternativa aos bancos de dados relacionais. Neste trabalho investigamos e comparamos a implementação do DW usando bancos de dados relacionais e NoSQL, considerando o *Star Schema Benchmark*. Experimentos foram conduzidos e os resultados obtidos sugerem que, para o cenário explorado, os banco de dados relacionais possuem resultado superior no processamento de consultas ao NoSQL orientado a documentos.

Palavras-chave: Data Warehouse; Business Intelligence; Processamento de Consultas; OLAP; NoSQL; Banco de Dados Relacional.

ABSTRACT

Over the last few decades, data management has become one of the most important activities in organizations, due to the growth in data generation around the world. Currently, the biggest challenge is to organize information and manipulate in order to assist in decision making. In this scenario, the Data Warehouse (DW) is an important component to support decisions. DW is a non-volatile database, subject-oriented, historical and massive, whose processing of analytic queries results in downtime high answer Techniques are used to improve the performance of processes query processing, among them are the use of data fragmentation, materialized views and indices. Furthermore, NoSQL is an emerging technology whose main features are the improvement of query processing and data storage, becoming an alternative to relational databases. In this work we investigate and compare the implementation of Dw using relational databases and NoSQL, considering the Star Benchmark Schema. Experiments were carried out and the results obtained suggest that, for the scenario explored, relational databases have a superior result in processing of queries to document-oriented NoSQL.

Keywords: Data Warehouse; Business Intelligence; Query Processing; OLAP; NoSQL; Relational Database.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

No decorrer dos últimos anos as organizações estão enfrentando desafios importantes no ambiente corporativo, tratando-se dos dados gerados por todo o mundo. Devido estes cenários, as informações/dados estão se tornando uma fonte de grande vantagem competitiva. Com isso adotam cada vez mais a inteligência de negócio, *Business Intelligence* (BI), para compreender e analisar informações para auxiliar à tomada de decisão estratégica. Decisões assertivas levam a processos melhores e são mais eficientes no ambiente de trabalho. O BI é uma condição importante onde os gestores, de todas as áreas, devem estar cientes e utilizam como uma fonte de vantagem competitiva(FOLEY; GUILLEMETTE, 2010).

Para o auxílio da elaboração de estratégias para a tomada de decisão, necessitam de consultas com tempo de resposta baixo. Porém, em geral, o processamento de consultas utiliza um grande volume de dados não integrados e oriundos de diversos provedores de informação, os quais se encontram armazenados em um *Data Warehouse* (CARNIEL et al., 2015). Um *Data Warehouse* (DW) incorpora esses dados para auxiliar positivamente na decisão tática, estratégica e operacional da organização (XU et al., 2007). Segundo (KIMBALL; ROSS, 2011) DW é uma base de dados histórica, volumosa, não volátil e orientada ao assunto.

Processar consultas com eficiência em um DW é um grande desafio devido ao grande volume de dados. Tais consultas processam diversas junções, agrupamentos, filtros e ordenações. Visando melhorar o desempenho de reposta das consultas **OLAP** (*Online Analytical Processing*), existem técnicas bem conhecidas como: fragmentação dos dados, visão materializada e estruturas de indexação (CARNIEL et al., 2015).

1.1 Problema e Motivação

O DW pode ser implementado em diferentes tecnologias de armazenamento de dados, sendo eles o SGBD Relacional, NoSQL e entre outros. Atualmente, com grandes volumes de dados é um desafio processar consultas eficientes em um DW. Uma vez que tais consultas necessitam realizar junções entre tabelas, agrupamentos, filtros e ordenações. Com isto existem dificuldades para mensurar ganhos relacionado a consultas nos bancos de dados.

A motivação deste trabalho de conclusão de curso é reproduzir o trabalho realizado em 2012 pelos autores Anderson Chaves Carniel, Aried de Aguiar Sá, Vinícius Henrique Porto Brisighello, Marcela Xavier Ribeiro, Renato Bueno, Ricardo Rodrigues Ciferri, intitulado como "*Query processing over data warehouse using relational databases and NoSQL*" (CARNIEL et al., 2012), para a linguagem de banco de dados atuais. Investigar e comparar o uso de modelos de dados relacional e NoSQL, utilizando as principais técnicas para otimização de processamento de consultas OLAP sobre DW.

A técnica experimental de avaliação de desempenho utilizada em aplicações e sistemas

de bancos de dados é composta principalmente da técnica de *benchmark*, que consiste em um conjunto de testes experimentais previamente definidos e posteriormente executados para obtenção de resultados de desempenho. Por consequência, neste trabalho é comparado ferramentas que seguem os modelos orientados a documentos com o modelo relacional, analisando o seu espaço de armazenamento e o tempo de processamento das consultas sobre o *Star Schema Benchmark* (SSB).

Isso se deve por questões que, durante os últimos anos as organizações estão cada vez mais competitivas em razão da melhora significativa na utilização dos dados, não somente coletadas internamente como informações externas.

1.2 Objetivos

O objetivo geral deste trabalho é realizar a análise de consultas OLAP sobre DW para identificar entre os modelos de dados relacional e NoSQL, qual tem melhor desempenho no processamento de consultas OLAP e um bom uso de armazenamento.

Os objetivos específicos deste trabalho são:

- Fazer estudo sobre *Business Intelligence*, Data Warehouse e consultas OLAP;
- Utilizar uma base de dados sintética, *Star Schema Benchmark*, para construção do DW com modelo NoSQL e SGBD Relacional;
- Analisar o processamento de consultas no DW e seu armazenamento;

1.3 Contribuições

As contribuições deste trabalho de conclusão são listadas abaixo:

- Durante a construção deste projeto foi realizado a transcrição das consultas OLAP do SSB para a versão atual do MongoDB, levando em considerações o trabalho de estudo. Entretanto na execução deste trabalho, ocorreram dificuldades com a linguagem do banco NoSQL. Devido as várias atualizações da linguagem no decorrer dos últimos anos. Há uma década atrás, utilizava-se o *mapReduce*, hoje em dia, na documentação do banco de dados orienta a aplicação de agregações por serem mais rápidas e o uso de código JavaScript com escopo para as funções *mapReduce* foi preterido desde a versão 4.2.1 (MONGODB, 2022).
- Ao longo do processo de coleta dos dados foi utilizado a linguagem de programação Python, visando uma coleta mais rápida, eficiente e fácil de manusear devido a familiaridade com tal linguagem. Anteriormente foi testado um *script bash* pelo WSL (Subsistema do Windows para Linux), porém esta técnica cria um ambiente de desenvolvimento parecido com uma máquina virtual, desta forma não foi possível realizar a comunicação com os bancos de dados instalados localmente.

- No decorrer do trabalho verificou-se a tentativa de avaliação de performance do banco de dados NoSQL orientado a grafos, mais precisamente do Neo4J, visto que é um banco de dados que sua utilização esta em crescimento nos últimos anos. Porém devido a grande demora na transformação da base de dados em razão do equipamento desatualizado e o tempo para conclusão do trabalho, decidiu-se por não seguir com o estudo para este trabalho.

1.4 Organização do Trabalho

Este trabalho está organizado nos seguintes capítulos:

- Capítulo 1: Neste primeiro capítulo introdutório é contextualizado o problema a ser abordado nas próximas seções, assim como os problemas e motivações, e os objetivos do trabalho.
- Capítulo 2: Este capítulo envolve o levantamento bibliográfico da pesquisa, no qual é tratado sobre *Data Warehouse*, *Star Schema Benchmark* (SSB) e visões fragmentadas verticalmente e materializadas.
- Capítulo 3: É discutido sobre o ambiente de execução das consultas e em sequência sobre as consultas SQL e NoSQL, neste caso sobre o banco de dados MongoDB.
- Capítulo 4: Apresenta e discute os principais resultados obtidos após a realização dos experimentos.
- Capítulo 5: São apresentadas as conclusões, assim como trabalhos futuros.

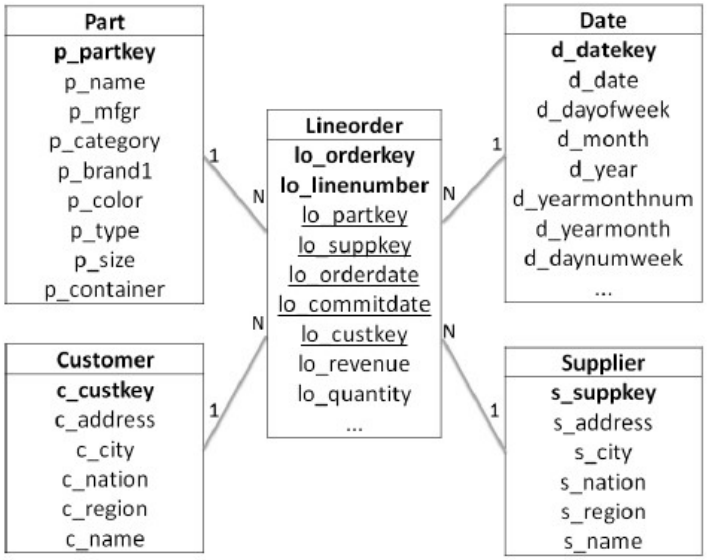
2 REVISÃO DE LITERATURA

Neste capítulo são abordados conceitos importantes para este trabalho, na seção 2.1 é retratado os fundamentos de um *Data Warehouse* e as consultas analíticas (OLAP). Já na seção 2.2 é apresentado sobre *benchmark* e o *Star Schema Benchmark* junto com seus graus de consultas. Por fim, é apresentado sobre visões fragmentadas e visões materializadas na seção 2.3.

2.1 Data Warehouse

Data Warehouses são grandes repositórios de dados analíticos e orientados por assunto integrados de várias fontes heterogêneos num longo período de tempo, sendo intrinsecamente volumosos. Um DW é representado por um hipercubo de dados multidimensional. Seu projeto lógico pode ser baseado no modelo relacional, utilizando o esquema estrela ou o esquema floco de neve. Compõe os dois esquemas, uma tabela de fatos e tabelas de dimensão (KIMBALL; ROSS, 2011). Na Figura 1 é apresentado um esquema estrela que tem a tabela de fatos *Lineorder* e as tabelas de dimensão *Supplier*, *Customer*, *Part* e *Date* (O'NEIL et al., 2009).

Figura 1 – Esquema Estrela de uma aplicação de varejo.



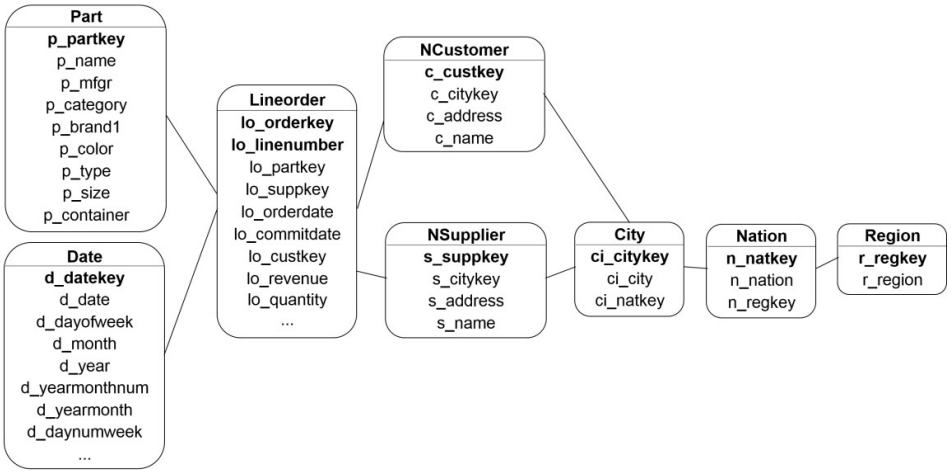
Fonte: (O'NEIL et al., 2009)

A tabela de fatos mantém os dados que são usados para calcular as métricas do negócio analisado, além de armazenar as chaves estrangeiras para registros das tabelas de dimensão. Já as tabelas de dimensão concedem características do negócio e seus atributos podem formar hierarquias (HARINARAYAN; RAJARAMAN; ULLMAN, 1996).

Hierarquias são coleções de níveis que permitem a agregação dos dados e consequentemente o processamento de consultas *drill-down* e *roll-up*, amplamente usadas em aplicações OLAP (XU et al., 2007). Por exemplo, a tabela de dimensão *Date* possui hierarquia entre os atributos d_year $\leq$ d_month $\leq$ d_daynuminmonth.

A única diferença entre o esquema floco de neve e o esquema estrela é a normalização das hierarquias contidas nas tabelas de dimensão (KIMBALL; ROSS, 2011). Consequentemente, haverá um número maior de junções. Na Figura 2 é apresentado o esquema floco de neve com as hierarquias *Supplier* e *Customer* normalizadas, uma vez que possuem atributos em comum. A hierarquia dessas dimensões é: *all* $\leq$ *region* $\leq$ *nation* $\leq$ *city* $\leq$ *address*.

Figura 2 – Esquema Floco de Neve de uma aplicação de varejo.



Fonte: (O'NEIL et al., 2009)

Consultas analíticas são processadas sobre um DW por ferramentas OLAP. Desta forma, fornecendo visões multidimensionais dos resultados e contribuindo com o planejamento estratégico das empresas (XU et al., 2007). Tais ferramentas geralmente acessam o DW utilizando sistemas de banco de dados relacionais para o processamento das consultas. Portanto, para a tomada de decisão estratégica, um fator de importância é o processamento eficiente de consultas OLAP visando reduzir o tempo de resposta das consultas (HARINARAYAN; RAJARAMAN; ULLMAN, 1996).

2.2 Star Schema Benchmark

Benchmarks são ferramentas que auxiliam responder perguntas como: "Qual é o melhor sistema em um determinado domínio?" ou, considerando o ambiente de *data warehousing*, "Qual é o melhor sistema de bancos de dados para operações OLAP?" (FOLKERTS et al., 2012). Tais perguntas são respondidas por meio de abordagens sistemáticas, baseando-se nas propriedades e restrições de cada sistema avaliado.

Portanto o objetivo de um *benchmark* é reportar comparativamente a qualidade e o desempenho de diferentes sistemas, levando em consideração as características e limitações de cada um. Isso serve como base para auxiliar uma empresa na escolha de um sistema para atender as suas necessidades de um determinado domínio, por exemplo (SCABORA, 2016).

Na literatura existem três principais *benchmarks* para avaliação de desempenho de DWs: TPC-H, TPC-DS e *Star Schema Benchmark* (SSB) (BARATA; BERNARDINO; FURTADO, 2014). Neste trabalho utilizaremos como base o SSB, que aplica o esquema estrela citado na Figura 1.

O SSB define ao todo 13 consultas, sendo algumas adaptadas do TPC-H enquanto outras foram criadas para validar o novo esquema. Essas consultas são divididas em quatro classes/grupos, a fim de realizar cobertura funcional e testar variações tanto de seletividade quanto de nível de agregação. Cada classe de consulta possui uma quantidade de *query* que se encaixa nas métricas do SSB.

As consultas da classe 1 (nomeado neste trabalho como: Q1) dispõem de apenas uma junção entre a tabela de fatos *Lineorder* e a tabela de dimensão *Date* através da soma total, sem dividir os dados em diferentes grupos. Enquanto as consultas de classes 2, 3 e 4 (no nosso caso: Q2, Q3 e Q4, respectivamente) agregam os dados por meio da soma, os quais são separados em grupos e originários da união da tabela de fatos *Lineorder* com as tabelas dimensões *Date*, *Part* e *Supplier*, no caso da classe 2. E para a classe 3, consiste na união da tabela fato com as tabelas dimensões *Date*, *Customer* e *Supplier* e na classe 4 é utilizada todas as tabelas do esquema.

A seguir é ilustrado a primeira consulta de cada classe do SSB, enquanto as demais são apresentadas nos apêndices (A e B). O código abaixo refere-se a primeira consulta da classe 1, sendo chamado neste trabalho como Q1.1. A diferença entre as consultas do grupo Q1 é somente os filtros, cláusula *WHERE*.

```
1 SELECT SUM(lo_extendedprice*lo_discount) AS revenue
2 FROM lineorder, date
3 WHERE lo_orderdate = d_datekey
4     AND d_year = [YEAR]
5     AND lo_discount BETWEEN [DISCOUNT] - 1 AND [DISCOUNT] + 1
6     AND lo_quantity < [QUANTITY];
```

Os colchetes aplicados nas consultas indicam argumentos (parâmetros) das consultas que permitem filtrar por características específicas desejadas. Por exemplo, no grupo Q1 existem 3 tipos de parâmetros a serem definidos: `[YEAR]`, `[DISCOUNT]` e `[QUANTITY]`.

Consultas do grupo Q2 tem restrição para duas dimensões e comparam a receita de algumas classes de produtos para fornecedores de uma determinada região. Estes são agrupados por classes de produtos mais restritivas e todos os anos de pedidos.

```
1 SELECT SUM(lo_revenue), d_year, p_brand1
2 FROM lineorder, date, part, supplier
3 WHERE lo_orderdate = d_datekey
```

```
    AND lo_partkey = p_partkey
    AND lo_suppkey = s_suppkey
    AND p_category = [CATEGORY]
    AND s_region = [REGION]
GROUP BY d_year, p_brand1
ORDER BY d_year, p_brand1;
```

Como citado anteriormente, as consultas do grupo Q3 tem restrições para três dimensões. As consultas deste grupo destinam-se a fornecer o volume de receita para transações de pedidos simples, por país do cliente e país do fornecedor e ano de uma determinada região, em um determinado período de tempo.

```
SELECT c_nation, s_nation, d_year, SUM(lo_revenue) AS revenue
FROM customer, lineorder, supplier, date
WHERE lo_custkey = c_custkey
    AND lo_suppkey = s_suppkey
    AND lo_orderdate = d_datekey
    AND c_region = [REGION]
    AND s_region = [REGION]
    AND d_year >= [YEAR] AND d_year <= [YEAR]
GROUP BY c_nation, s_nation, d_year
ORDER BY d_year asc, revenue DESC;
```

E por fim as consultas do grupo Q4 que representam uma sequência "e se" do tipo OLAP. Começando com um *GROUP BY* em duas dimensões e restrições bastante fracas em três dimensões, e medimos o lucro agregado, medido como: (lo_revenue - lo_supplycost).

```
SELECT d_year, c_nation, SUM(lo_revenue - lo_supplycost) AS profit
FROM date, 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 = [REGION]
    AND s_region = [REGION]
    AND (p_mfgr = [MFGR] OR p_mfgr = [MFGR])
GROUP BY d_year, c_nation
ORDER BY d_year, c_nation;
```

Na Tabela 1 é mostrada a complexidade de cada grupo de consultas. No qual cada grupo de consultas tem números específicos de junções e atributos, bem como agregações, filtros e ordenações.

Identifica-se que para o grupo Q1 existe somente uma junção entre as tabelas, utiliza-se 3 filtros, 4 atributos e não engloba agregação e ordenação. Enquanto para os grupos Q2 e Q3, a diferença entre os dois é somente na quantidade de filtros e atributos utilizados, onde Q3 possui 3 filtros e 7 atributos enquanto Q2 tem 2 filtros e 5 atributos. O grupo de consultas Q4 é o que possui a maior complexidade entre todos, contendo 4 junções de tabelas, 3 filtros, 7 atributos e possuindo agregação e ordenação nas consultas.

Tabela 1 – Complexidade de cada grupo de consultas do SSB

	Q1	Q2	Q3	Q4
Junções	1	3	3	4
Filtros	3	2	3	3
Agregação?	Não	Sim	Sim	Sim
Ordenação?	Não	Sim	Sim	Sim
Atributos?	4	5	7	7

Fonte: Carniel et al. (2012)

2.3 Visões Fragmentadas e Materializadas

O processamento de consultas em DW envolve junções, agrupamentos, filtros e ordenações sobre um grande volume de dados. A forma mais custosa para se processar uma consulta em DW é a junção estrela (JE) que computa todas as junções, agrupamentos, filtros e ordenações envolvidas (CARNIEL et al., 2012). Desta forma torna-se um método com menor recomendação para processamento de consultas eficientes sobre um DW. Porém, existem outros métodos para melhorar o desempenho destas consultas, sendo elas: fragmentação de dados (GOLFARELLI; MAIO; RIZZI, 2000), visão materializada (BAIKOUSI; VASSILIADIS, 2009) e estruturas de indexação (O'NEIL; GRAEFE, 1995).

Para a construção de uma visão fragmentada verticalmente (VFV), utiliza-se a técnica de fragmentação dos dados, a qual mantém uma cópia dos dados mantidos no DW em uma tabela relacional separada (GOLFARELLI; MAIO; RIZZI, 2000). Tal tabela mantém os dados de uma determinada junção entre tabelas de fatos e de dimensão. Assim, mantendo o conjunto mínimo de atributos necessários para responder a um conjunto de consultas ou um *benchmark*. Desta forma, consultas sobre a VFV dispensam o uso de junções, apenas computando seleções, agrupamentos e ordenações sobre os dados.

Já uma visão materializada (VM), é uma tabela relacional que mantém os dados agregados de um subconjunto de atributos de uma determinada junção entre tabelas de fatos e de dimensão (BAIKOUSI; VASSILIADIS, 2009). Uma VM mantém o conjunto mínimo de atributos, processando previamente agrupamentos e resultados das funções de agregação sobre as medidas. Portanto, uma VM ocupa menos espaço de armazenamento que uma VFV, pois reduz os dados armazenados devido ao seu agrupamento dos dados (CARNIEL et al., 2012).

3 Preparação da Avaliação Experimental

Neste capítulo são descritas as definições de comandos para os modelos de banco de dados propostos e os métodos de coleta de dados. Todos os demais comandos são apresentados no apêndice **??** e **??**.

3.1 Definição de Comandos SQL para o PostgreSQL

Antes de realizar o estudo de performance do SSB sobre o DW, é necessário efetuar as definições dos comandos para criação das visões fragmentadas e visões materializadas no modelo de banco de dados PostgreSQL. Com isso, foram criadas novas tabelas seguindo os requisitos de cada classe e suas respectivas consultas.

A seguir é ilustrado um comando de cada classe do SSB, tais comandos foram aplicados nas criações das visões fragmentadas na base de dados. Os demais comandos são evidenciados no apêndice **??**.

As criações das VFVs para o grupo de consultas Q1 definem as junções entre a tabela de fatos *Lineorder* com a tabela de dimensão *Date*.

VFV Q1.1:

```
1 CREATE TABLE vfv_q1_1 AS (
2  SELECT lo_extendedprice, lo_discount, lo_quantity, d_year
3  FROM lineorder, date
4  WHERE lo_orderdate = d_datekey
5 );
```

Com base na estrutura demonstrada anteriormente, é realizado os comandos para criação das VFVs dos grupos de consultas Q2, Q3 e Q4. Portanto, é descrito a seguir os códigos SQL que correspondem a criação das VFVs do grupo de consulta Q2.

VFV Q2.1:

```
1 CREATE TABLE vfv_q2_1 AS (
2  SELECT lo_revenue, d_year, p_brand1, p_category, s_region
3  FROM date, part, lineorder, supplier
4  WHERE  lo_suppkey = s_suppkey
5     AND lo_orderdate = d_datekey
6     AND lo_partkey = p_partkey
7 );
```

Para a criação das VFVs do grupo Q3, é definido as junções entre a tabela de fatos *Lineorder* com as tabelas de dimensão *Customer*, *Supplier* e *Date*.

VFV Q3.1:

```
1 CREATE TABLE vfv_q3_1 AS (
2  SELECT c_nation, s_nation, c_region, s_region, d_year, lo_revenue
3  FROM customer, lineorder, supplier, date
```

```
4  WHERE  lo_custkey = c_custkey
5     AND lo_suppkey = s_suppkey
6     AND lo_orderdate = d_datekey
7 );
```

E por último, os comandos de criação para o grupo Q4. A qual realiza a junção entre a tabela de fatos com todas as tabelas de dimensão:

VFV Q4.1:

```
1 CREATE TABLE vfv_q4_1 AS (
2  SELECT d_year, c_nation, lo_revenue, lo_supplycost, c_region, s_region,
      p_mfgr
3  FROM date, customer, supplier, part, lineorder
4  WHERE  lo_custkey = c_custkey
5     AND lo_suppkey = s_suppkey
6     AND lo_partkey = p_partkey
7     AND lo_orderdate = d_datekey
8 );
```

A partir da criação das visões na base de dados, o próximo passo é elaborar os comandos para as consultas sobre estas VFVs, englobando todas as 13 consultas, separando-as pelas classes definidas no SSB.

Para todas as consultas que são realizadas sobre as VFVs é necessário efetuar somente os filtros, agrupamentos, ordenação e função de agregação. Conforme demonstrado abaixo nas primeiras consultas de cada classe abaixo.

Consulta VFV Q1.1:

```
1 SELECT SUM(lo_extendedprice*lo_discount) AS revenue
2  FROM vfv_q1_1
3  WHERE d_year = 1993
4     AND lo_discount BETWEEN 1 AND 3
5     AND lo_quantity < 25;
```

Consulta VFV Q2.1:

```
1 SELECT SUM(lo_revenue), d_year, p_brand1
2  FROM vfv_q2_1
3  WHERE p_category = 'MFGR#12'
4     AND s_region = 'AMERICA'
5  GROUP BY d_year, p_brand1
6  ORDER BY d_year, p_brand1;
```

Consulta VFV Q3.1:

```
1 SELECT c_nation, s_nation, d_year, SUM(lo_revenue) AS revenue
2  FROM vfv_q3_1
3  WHERE c_region = 'ASIA'
4     AND s_region = 'ASIA'
5     AND d_year >= 1992
6     AND d_year <= 1997
```

```
7  GROUP BY c_nation, s_nation, d_year
8  ORDER BY d_year ASC, revenue DESC;
```

Consulta VFV Q4.1:

```
1 SELECT d_year, c_nation, SUM(lo_revenue - lo_supplycost) AS profit
2  FROM vfv_q4_1
3  WHERE c_region = 'AMERICA'
4     AND s_region = 'AMERICA'
5     AND (p_mfgr = 'MFGR#1' OR p_mfgr = 'MFGR#2')
6  GROUP BY d_year, c_nation
7  ORDER BY d_year, c_nation;
```

Já para a criação das VMs e suas respectivas consultas segue as mesmas preparações realizadas para o método VFV. Respeitando os requisitos estabelecidos pelo SSB. Todos os comandos são expostos no apêndice **??**.

3.2 Definição de Comandos no MongoDB

Com a finalidade de determinar os comandos para o banco de dados do MongoDB, utiliza-se os comandos comandos gerados anteriormente para a linguagem SQL como base. Assim sendo, os comandos/consultas foram reescritos seguindo as sintaxes do MongoDB.

Para a utilização do modelo de banco de dados MongoDB, a tabela de fatos e as tabelas de dimensão se transformaram em uma única *collection* devido necessidade da linguagem. Ou seja, na linguagem SQL consistia em 5 tabelas (*Lineorder*, *Customer*, *Date*, *Supplier* e *Part*) e para a linguagem NoSQL, orientada a documentos, tem somente a *collection Lineorder* contendo todas as informações.

A seguir é apresentado um exemplo de uma consulta para cada classe do SSB, utilizando o método junção estrela para o banco de dados MongoDB. O colchete aplicado nas consultas indica a informação que deve ser manipulada conforme necessidade ao nome da *collection* definida na base de dados.

Consulta Q1.1:

```
1 db.[COLLECTION].aggregate([{
2  $match:{
3   'date.d_year': {$eq: 1993},
4   lo_quantity: {$lt: 25},
5   lo_discount: {$gte: 1, $lte: 3}
6  }
7 },{
8  $group:{
9   _id: null,
10  revenue:{
11    $sum: { $multiply: ['$lo_extendedprice', '$lo_discount']}}
12 }
13 }])
```

O próximo comando é referente a consulta Q2.1:

```
db.[COLLECTION].aggregate([{
 $match:{
  'part.p_category': 'MFGR#12',
  'supplier.0.s_region': {$eq: 'AMERICA'}
 }
},{
 $group: {
  _id:{
    d_year: '$date.d_year',
    p_brand1: '$part.p_brand1'
  },
  revenue: {
   $sum: '$lo_revenue'
  }
 }
}])
```

Consulta Q3.1:

```
db.[COLLECTION].aggregate([{
 $match: {
  'date.d_year': {$in: [1992, 1997]},
  'supplier.s_region': {$eq: 'ASIA'},
  'customer.c_region': 'ASIA'
 }
},{
 $group: {
  _id: {
    c_nation: '$customer.c_nation',
    s_nation: '$supplier.s_nation',
    d_year: '$date.d_year'
  },
  revenue: {
   $sum: '$lo_revenue'
  }
},{
 $sort: {
  d_year: 1,
  revenue: -1
 }
}])
```

Consulta Q4.1:

```
db.[COLLECTION].aggregate([{
 $match:{
  'customer.c_region': {$eq: 'AMERICA'},
  'supplier.s_region': {$eq: 'AMERICA'}
```

```
 }
},{
 $match: {
  $or: [{'part.p_mfgr': {$eq: 'MFGR#1'}},
        {'part.p_mfgr': {$eq: 'MFGR#2'}}]
 }
},{
 $group: {
  _id: {
    d_year: '$date.d_year',
    c_nation: '$customer.c_nation'},
  profit: {
    $sum: {$subtract: ['$lo_revenue', '$lo_supplycost']}}
 }
}])
```

Após as consultas das junções estrelas, sucedeu a criação de novas *collections* para as Visões Fragmentadas e Materializadas devido necessidades de modelagem do banco de dados. Neste estágio foi gerado uma VFV para cada consulta do SSB, com igualdade aos processos do PostgreSQL.

Os comandos a seguir apresentam um exemplo de criação das *collections* por grupo de consulta e método (VFV ou VM), começando pelas Visões Fragmentadas.

VFV Q1.1:

```
db.[COLLECTION].aggregate([{
 $project: {
  lo_extendedprice: 1,
  lo_discount: 1,
  lo_quantity: 1,
  'date.d_year': 1
 }
},{
 $merge: {
  into: 'vfv_q1_1',
  whenMatched: 'replace',
  whenNotMatched: 'insert'
 }
}])
```

VFV Q2.1:

```
db.[COLLECTION].aggregate([{
 $project: {
  lo_revenue: 1,
  'date.d_year': 1,
  'part.p_brand1': 1,
  'part.p_category': 1,
  'supplier.s_region': 1
```

```
 }
},{
 $merge: {
  into: 'vfv_q2_1',
  whenMatched: 'replace',
  whenNotMatched: 'insert'
 }
}])
```

VFV Q3.1:

```
db.[COLLECTION].aggregate([{
 $project: {
  'customer.c_nation': 1,
  'supplier.s_nation': 1,
  'customer.c_region': 1,
  'supplier.s_region': 1,
  'date.d_year': 1,
  lo_revenue: 1
 }
}, {
 $merge: {
  into: 'vfv_q3_1',
  whenMatched: 'replace',
  whenNotMatched: 'insert'
 }
}])
```

VFV Q4.1:

```
db.[COLLECTION].aggregate([{
 $project: {
  'date.d_year': 1,
  'customer.c_nation': 1,
  lo_revenue: 1,
  lo_supplycost: 1,
  'customer.c_region': 1,
  'supplier.s_region': 1,
  'part.p_mfgr': 1
 }
}, {
 $merge: {
  into: 'vfv_q4_1',
  whenMatched: 'replace',
  whenNotMatched: 'insert'
 }
}])
```

Após a criação das collections para as VFVs, foi reestruturado os comandos para as VMs, seguindo a mesma lógica mas respeitando as junções de cada grupo.

VM Q1.1:

```
db.[COLLECTION].aggregate([{
 '$project': {
  'date.d_year': 1,
  'lo_discount': 1,
  'lo_quantity': 1,
  'lo_extendedprice': 1,
  'lo_revenue': 1,
  'customer.c_city': 1
 }
},{
 '$group': {
  '_id':{
      'd_year': '$date.d_year',
      'lo_discount': '$lo_discount',
      'lo_quantity': '$lo_quantity'
   },
   'revenue':{
      '$sum': '$lo_revenue'
   }
  }
},{
  '$merge': {
    'into': 'vm_q1_1',
    'whenMatched': 'replace',
    'whenNotMatched': 'insert'
    }
}])
```

VM Q2.2:

```
db.[COLLECTION].aggregate([{
 $project: {
  lo_revenue: 1,
  'date.d_year': 1,
  'part.p_brand1': 1,
  'part.p_category': 1,
  'supplier.s_region': 1
 }
},{
 $group: {
  _id: {
   d_year: '$date.d_year',
   p_brand1: '$part.p_brand1',
   p_category: '$part.p_category',
   s_region: '$supplier.s_region'
  },
  revenue: {
```

```
    $sum: '$lo_revenue'
   }
  }
},{
 $merge: {
  into: 'vm_q2_2',
  whenMatched: 'replace',
  whenNotMatched: 'insert'
 }
}])
```

VM Q3.1:

```
db.[COLLECTION].aggregate([{
 $project: {
  lo_revenue: 1,
  'date.d_year': 1,
  'supplier.s_region': 1,
  'supplier.s_nation': 1,
  'customer.c_nation': 1,
  'customer.c_region': 1
 }
}, {
 $group: {
  _id: {
   d_year: '$date.d_year',
   s_region: '$supplier.s_region',
   c_region: '$customer.c_region',
   s_nation: '$supplier.s_nation',
   c_nation: '$customer.c_nation'
  },
  revenue: {
   $sum: '$lo_revenue'
  }
 }
}, {
 $merge: {
  into: 'vm_q3_1',
  whenMatched: 'replace',
  whenNotMatched: 'insert'
 }
}])
```

VM Q4.1:

```
db.[COLLECTION].aggregate([{
 $project: {
  lo_revenue: 1,
  lo_supplycost: 1,
  'date.d_year': 1,
```

```
  'supplier.s_region': 1,
  'customer.c_nation': 1,
  'customer.c_region': 1,
  'part.p_mfgr': 1
 }
}, {
 $group: {
  _id: {
   d_year: '$date.d_year',
   c_nation: '$customer.c_nation'
  },
  profit: {
   $sum: { $subtract: ['$lo_revenue', '$lo_supplycost'] }
  }
 }
}, {
 $merge: {
  into: 'vm_q4_1',
  whenMatched: 'replace',
  whenNotMatched: 'insert'
 }
}])
```

Seguido da criação das Visões no banco de dados, é realizado as consultas conforme *benchmark* utilizado neste trabalho. A seguir é demonstrado um exemplo de consulta a classe 3, sendo a primeira desta classe (Q3.1).

Consulta VFV Q3.1:

```
db.vfv_q3_1.aggregate([{
 $match:{
  'date.d_year':{$in:[1992, 1997]},
  'supplier.s_region':{$eq: 'ASIA'},
  'customer.c_region':{$eq: 'ASIA'}
 }
},{
 $group:{
  _id:{
   c_nation: '$customer.c_nation',
   s_nation: '$supplier.s_nation',
   d_year: '$date.d_year'},
  revenue: {
   $sum: '$lo_revenue'
  }
 }
},{
 $sort:{
  d_year: 1,
  revenue: -1,
```

```
21   _id: 1
22  }
23 }]
```

Consulta VM Q3.1:

```
1 db.vm_q3_1.aggregate([{
2  $match:{
3   '_id.d_year':{ $in: [1992, 1997]},
4   '_id.c_region':{ $eq: 'ASIA'},
5   '_id.s_region':{$eq: 'ASIA'}
6  }
7 },{
8  $project:{
9   '_id.c_region': 0,
10  '_id.s_region': 0
11 }
12 },{
13 $sort:{
14  d_year: 1,
15  revenue: -1,
16  _id: 1
17 }
18 }]
```

Anteriormente nesta seção foi demonstrado somente um exemplo de cada consulta ou criação das visões para cada grupo do SSB. Portanto no apêndice **??** são expostos todos os comandos utilizados neste trabalho para o banco de dados MongoDB.

3.3 Execução dos Experimentos

Os experimentos foram realizados localmente com a finalidade de inibir a latência da rede e para a coleta do dados foi desenvolvido um *script* em Python.

Este *script* foi desenvolvido em duas partes. Inicialmente é realizado a conexão com os bancos de dados e em seguida executa cada consulta separadamente, respeitando a linguagem do banco de dados, sua classe, método e consulta junto ao SSB. Seguidamente é registrado o seu tempo que levou para a execução dos comandos definidos e armazenando seu resultado.

Cada consulta do SSB foi executada cinco vezes. Com isso, é medido o tempo médio de execução em segundos de cada uma das 13 consultas do SSB. Para cada ciclo completo de execução do *script*, ou seja, quando finalizar todas as consultas e capturar todas as informações requisitadas, o equipamento é reiniciado com a finalidade de zerar o cache da máquina. Somente após esse procedimento que um novo ciclo é iniciado para a coleta de dados.

Na segunda parte é desenvolvido os tratamentos das informações coletadas, do passo anterior, e realizado as plotagens dos gráficos. O estudo sobre os resultados obtidos são demonstrados na Seção 4.

A plataforma de hardware e software utilizada para execução dos testes foi um computador com um processador Intel(R) Core(TM) i5-4210 com frequência de 1,70GHz, disco rígido SanDisk X600 *Solid State Drive* (SSD) de 500 GB e 8 GB de memória RAM. O sistema operacional foi Windows 10 com a versão 21H1 (Compilação do Sistema Operacional 19043.1706) e instalados os seguintes softwares: PostgreSQL 13, Studio 3T 2022.4.1, MongoDB Compass 1.32.2, Java JDK 8.0.2210.11 e Jupyter Notebook. Já para a execução do *script* em Python (Versão 3.8.3), utilizou-se as seguinte bibliotecas: pandas, matplotlib, numpy, psycopg2 e pymongo.

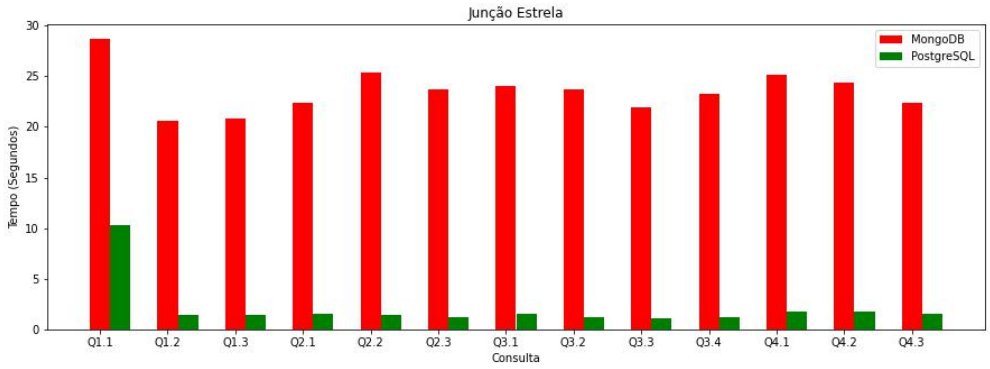
4 RESULTADOS

Neste capítulo serão apresentados os resultados das análises de desempenho de cada banco de dados através dos dados coletados. Detalhes da coleta são expostos na Seção 3.3.

Nos experimentos considerou-se os dados adquiridos em ambos os bancos de dados de estudo, no caso deste trabalho, PostgreSQL para a linguagem SQL e MongoDB para NoSQL. Durante a etapa de obtenção dos dados, 5 arquivos em planilhas excel foram criados, um para cada execução do *script* que coleta as informações das consultas do SSB.

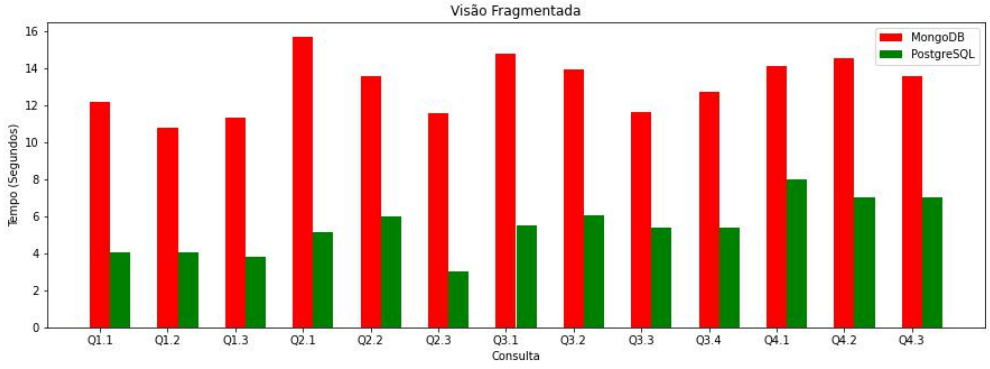
Porém, a partir da coleta destas informações, foram geradas as Figuras 3, Figuras 4 e Figuras 5 que demonstram o tempo médio em que cada consulta levou para obter o resultado dos métodos avaliados nestes trabalho, ou seja, junção estrela, visão fragmentada e visão materializada. O tempo médio das consultas foi obtido através do resultado da Seção 3.3.

Figura 3 – Consultas da Junção Estrela do SSB nos Banco de Dados.



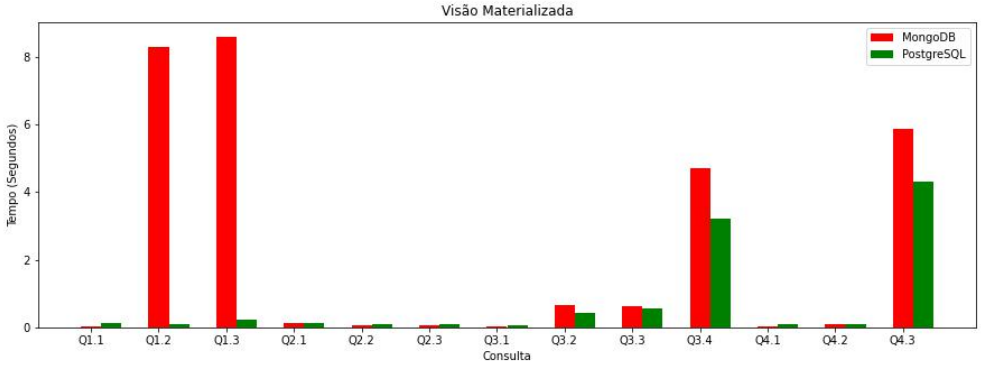
Fonte: Autoria própria.

Figura 4 – Consultas da Visão Fragmentada do SSB nos Banco de Dados.



Fonte: Autoria própria.

Figura 5 – Consultas da Visão Materializada do SSB nos Banco de Dados.



Fonte: Autoria própria.

A partir da análise da Figura 3, no qual é demonstrado as consultas do tipo Junção Estrela, verifica-se que o banco de dados MongoDB teve um tempo bem superior para processar as consultas do que o PostgreSQL. Entretanto, nota-se que na Figura 4 as consultas do MongoDB com o uso de documentos incorporados na *collection Lineorder* e a minimização na quantidade de atributos em uma coleção auxiliou no processamento das consultas OLAP, desta forma diminuindo o seu tempo de execução. Anexado a Tabela 2 é possível notar que a diferença no processamento das consultas do banco de dados MongoDB teve uma queda considerável com as incorporações e minimizações dos atributos. Seu melhor desempenho foi com a utilização da Visão Materializada, em que chegou mais próximo dos resultados do PostegreSQL.

Verifica-se na Figura 4, no qual é demonstrado as consultas do tipo Visão Fragmentada, que as consultas do banco de dados PostgreSQL obtiveram um aumento considerável, dado a importância com as consultas realizadas da Junção Estrela (Figura 3) e Visão Materializada (Figura 5). Nota-se que o tempo de processamento foi maior, chegando a levar um tempo maior que 100% em alguns casos.

Quanto a Tabela 2, é possível observar que todas as médias das consultas realizadas pelo banco de dados relacional foram inferiores ao MongoDB, independente do método utilizado (Junção Estrela, Visão Fragmentada ou Visão Materializada). Além disso, nota-se que para a Visão Fragmentada o tempo decorrido foi maior que os outros dois métodos. Entretanto, na Figura 6 podemos perceber que o tamanho em disco utilizado por essa visão é a maior entre os demais métodos.

Na Tabela 3 é demonstrado a média de tempo de execução para cada banco de dados, ou seja, foi considerado todas as consultas realizadas para trazer o resultado esperado. Porém, é possível analisar que a diferença média para realizar a execução dos comandos é considerável alta, pois a diferença entre os dois bancos de dados é aproximadamente de 10 segundos.

O alto tempo de processamento das consultas para o MongoDB ocorre devido o banco de dados ser não estruturado, ou *schema less*, assim ele não força uma estrutura definida para

Tabela 2 – Média por tipo de consulta SSB nos Banco de Dados

	Método	Tempo Decorrido (Segundos)
PostgreSQL	Junção Estrela	2.100436
	Visão Fragmentada	5.441393
	Visão Materializada	0.734712
MongoDB	Junção Estrela	23.556325
	Visão Fragmentada	13.136207
	Visão Materializada	2.246791

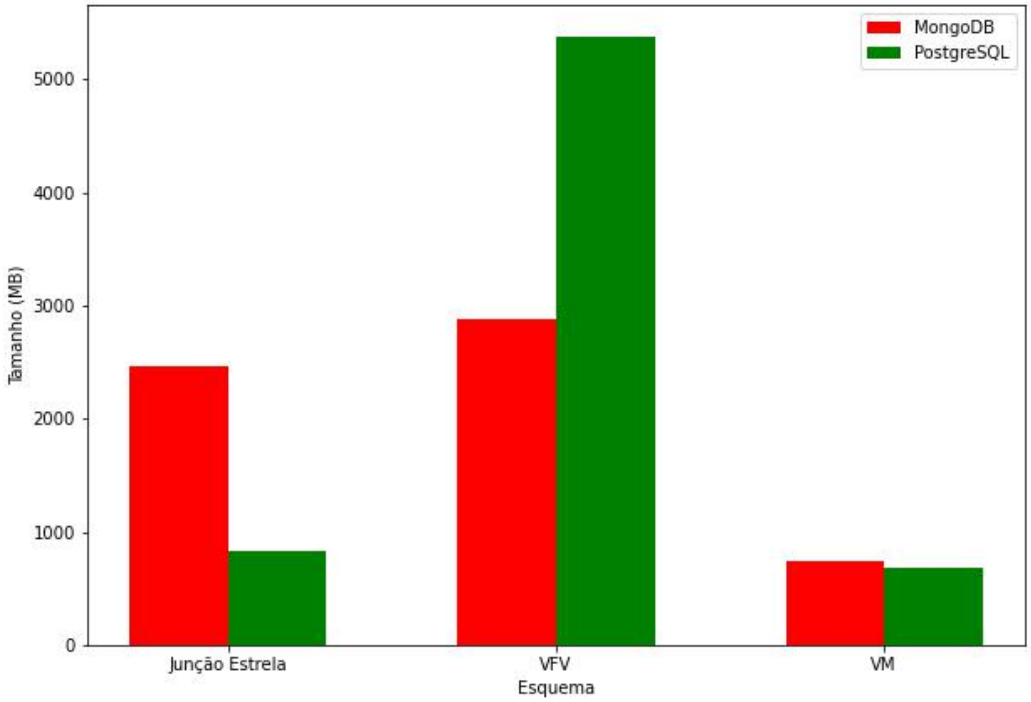
Fonte: Autoria própria.

Tabela 3 – Média das consultas SSB nos Banco de Dados

	Média Consultas (Segundos)
PostgreSQL	2.758847
MongoDB	12.979774

Fonte: Autoria própria.

Figura 6 – Tamanho em MB utilizado em disco de cada esquema do SSB.



Fonte: Autoria própria.

os documentos. Desse modo, uma coleção pode possuir vários documentos diferentes, cada um contendo campos distintos. Na contra-mão esta o banco de dados PostgreSQL, que é um banco de dados estruturado. Desta forma tem um padrão pré definido, uma estrutura bem definida e rígida.

Tabela 4 – Tamanho do banco de dados em disco (em MB).

	PostgreSQL	MongoDB
Junção Estrela	834.05	2467.84
Visão Fragmentada	5379.00	2874.49
Visão Materializada	682.61	744.31
Total	6895.66	6086.64

Fonte: Autoria própria.

5 CONCLUSÃO

Este trabalho investigou um modelo NoSQL e relacional, utilizando técnicas conhecidas para minimizar o tempo de respostas no processamento de consultas OLAP sobre DW. Na investigação experimental baseada no *Star Schema Benchmark*, foi utilizado a base de dados contendo volumes de dados sintéticos distintos. Investigou-se o tempo médio de resposta das consultas em relação ao custo de armazenamento. O objetivo foi identificar um modelo de dados (NoSQL ou relacional) que proporcionasse o melhor desempenho no processamento de consultas OLAP aliado a um bom uso do espaço de armazenamento.

Embora os bancos de dados NoSQL vem crescendo no mercado, o modelo relacional ainda se mostra bastante eficiente em seu desempenho. Com sua conclusão, fica evidente que embora o NoSQL tenha sido criado para minimizar alguns problemas de performance, o modelo relacional ainda sim é um modelo com desempenho satisfatório, e na maioria das ocasiões até superior, levando em conta o cenário estabelecido e os bancos de dados escolhidos.

Em relação aos custos de armazenamento, o modelo NoSQL orientado a documentos do MongoDB apresentou os melhores resultados. Porém não ofereceu reduções nos tempos de resposta no processamento das consultas. O PostgreSQL obteve o maior custo de armazenamento, comparado ao modelo do MongoDB, mesmo a diferença sendo pequena entre eles.

5.1 Trabalhos Futuros

Apesar dos resultados positivos, o trabalho tem algumas limitações que refletem às escolhas feitas para a condução dos experimentos. Como principais limitações e soluções propostas para possíveis melhorias do trabalho, é possível pontuar os seguintes itens:

- Utilizar uma base de dados com uma quantidade maior de dados, em nosso caso foi utilizado uma base com 6 milhões de tuplas, para trabalhos futuros podemos utilizar uma com 60 milhões ou mais.
- Realizar avaliação aos outros modelos do NoSQL, sendo eles: orientado a grafos, chave-valor, orientado a serviços.
- Além da avaliação da performance sobre consultas OLAP, realizar também estudo sobre inserção, alteração e exclusão.
- Realizar avaliações com tecnologias novas que vão emergindo no mercado.
- Estudo sobre tamanho das bases em cada banco de dados utilizado.
- Utilizar equipamentos mais atuais para avaliação da performance.

Referências

BAIKOUSI, E.; VASSILIADIS, P. View usability and safety for the answering of top-k queries via materialized views. In: **Proceedings of the ACM twelfth international workshop on Data warehousing and OLAP**. [S.l.: s.n.], 2009. p. 97–104. Citado na página 16.

BARATA, M.; BERNARDINO, J.; FURTADO, P. Survey on big data and decision support benchmarks. In: SPRINGER. **International Conference on Database and Expert Systems Applications**. [S.l.], 2014. p. 174–182. Citado na página 14.

CARNIEL, A. C. et al. Query processing over data warehouse using relational databases and nosql. In: IEEE. **2012 XXXVIII Conferencia Latinoamericana En Informatica (CLEI)**. [S.l.], 2012. p. 1–9. Citado 2 vezes nas páginas 9 e 16.

CARNIEL, A. C. et al. Query processing over data warehouse using relational databases and nosql. v. 1, n. 3, p. 2, 2015. Citado na página 9.

FOLEY, E.; GUILLEMETTE, M. **What is Business Intelligence?** [S.l.]: International Journal of Business Intelligence Research (IJBIR), 2010. Citado na página 9.

FOLKERTS, E. et al. Benchmarking in the cloud: What it should, can, and cannot be. In: SPRINGER. **Technology Conference on Performance Evaluation and Benchmarking**. [S.l.], 2012. p. 173–188. Citado na página 13.

GOLFARELLI, M.; MAIO, D.; RIZZI, S. Applying vertical fragmentation techniques in logical design of multidimensional databases. In: SPRINGER. **International Conference on Data Warehousing and Knowledge Discovery**. [S.l.], 2000. p. 11–23. Citado na página 16.

HARINARAYAN, V.; RAJARAMAN, A.; ULLMAN, J. D. Implementing data cubes efficiently. **Acm Sigmod Record**, ACM New York, NY, USA, v. 25, n. 2, p. 205–216, 1996. Citado 2 vezes nas páginas 12 e 13.

KIMBALL, R.; ROSS, M. **The data warehouse toolkit: the complete guide to dimensional modeling**. [S.l.]: John Wiley & Sons, 2011. Citado 3 vezes nas páginas 9, 12 e 13.

MONGODB. **Map-Reduce**. 2022. Disponível em: <https://www.mongodb.com/docs/manual/core/map-reduce/>. Acesso em: 17 de Outubro de 2022. Citado na página 10.

O'NEIL, P.; GRAEFE, G. Multi-table joins through bitmapped join indices. **ACM SIGMOD Record**, ACM New York, NY, USA, v. 24, n. 3, p. 8–11, 1995. Citado na página 16.

O'NEIL, P. et al. The star schema benchmark and augmented fact table indexing. In: SPRINGER. **Technology Conference on Performance Evaluation and Benchmarking**. [S.l.], 2009. p. 237–252. Citado 2 vezes nas páginas 12 e 13.

SCABORA, L. d. C. **Avaliação do Star Schema Benchmark aplicado a bancos de dados NoSQL distribuídos e orientados a colunas**. Tese (Doutorado) — Universidade de São Paulo, 2016. Citado na página 14.

XU, L. et al. Research on business intelligence in enterprise computing environment. In: IEEE. **2007 IEEE International Conference on Systems, Man and Cybernetics**. [S.l.], 2007. p. 3270–3275. Citado 2 vezes nas páginas 9 e 13.

APÊNDICE A - Comandos no PostgreSQL

Neste apêndice é abordado todos os comandos realizados para criação das visões, sendo ela fragmentada ou materializada, e também suas respectivas consultas conforme comentado durante o trabalho.

A seguir é demonstrado todos os comandos para criação das Visões Fragmentadas Verticalmente (VFVs): VFV Q1.1:

```
CREATE TABLE vfv_q1_1 AS (
 SELECT lo_extendedprice, lo_discount, lo_quantity, d_year
 FROM lineorder, date
 WHERE lo_orderdate = d_datekey );
```

VFV Q1.2:

```
CREATE TABLE vfv_q1_2 AS (
 SELECT lo_extendedprice, lo_discount, lo_quantity, d_yearmonthnum
 FROM lineorder, date
 WHERE lo_orderdate = d_datekey );
```

VFV Q1.3:

```
CREATE TABLE vfv_q1_3 AS (
 SELECT lo_extendedprice, lo_discount, lo_quantity, d_weeknuminyear,
   d_year
 FROM lineorder, date
 WHERE lo_orderdate = d_datekey );
```

VFV Q2.1:

```
CREATE TABLE vfv_q2_1 AS (
 SELECT lo_revenue, d_year, p_brand1, p_category, s_region
 FROM date, part, lineorder, supplier
 WHERE  lo_suppkey = s_suppkey AND
        lo_orderdate = d_datekey AND
        lo_partkey = p_partkey );
```

VFV Q2.2:

```
CREATE TABLE vfv_q2_2 AS (
 SELECT lo_revenue, d_year, p_brand1, s_region
 FROM date, part, lineorder, supplier
 WHERE  lo_suppkey = s_suppkey AND
        lo_orderdate = d_datekey AND
        lo_partkey = p_partkey );
```

VFV Q2.3:

```
CREATE TABLE vfv_q2_3 AS (
 SELECT lo_revenue, d_year, p_brand1, s_region
 FROM date, part, lineorder, supplier
 WHERE  lo_suppkey = s_suppkey AND
        lo_orderdate = d_datekey AND
        lo_partkey = p_partkey );
```

VFV Q3.1:

```
CREATE TABLE vfv_q3_1 AS (
 SELECT c_nation, s_nation, c_region, s_region, d_year, lo_revenue
 FROM customer, lineorder, supplier, date
 WHERE  lo_custkey = c_custkey AND
        lo_suppkey = s_suppkey AND
        lo_orderdate = d_datekey );
```

VFV Q3.2:

```
CREATE TABLE vfv_q3_2 AS (
 SELECT c_city, s_city, c_nation, s_nation, d_year, lo_revenue
 FROM customer, lineorder, supplier, date
 WHERE  lo_custkey = c_custkey AND
        lo_suppkey = s_suppkey AND
        lo_orderdate = d_datekey );
```

VFV Q3.3:

```
CREATE TABLE vfv_q3_3 AS (
 SELECT c_city, s_city, d_year, lo_revenue
 FROM customer, lineorder, supplier, date
 WHERE  lo_custkey = c_custkey AND
        lo_suppkey = s_suppkey AND
        lo_orderdate = d_datekey );
```

VFV Q3.4:

```
CREATE TABLE vfv_q3_4 AS (
 SELECT c_city, s_city, d_year, d_yearmonth, lo_revenue
 FROM customer, lineorder, supplier, date
 WHERE  lo_custkey = c_custkey AND
        lo_suppkey = s_suppkey AND
        lo_orderdate = d_datekey );
```

VFV Q4.1:

```
CREATE TABLE vfv_q4_1 AS (
 SELECT d_year, c_nation, lo_revenue, lo_supplycost, c_region, s_region,
    p_mfgr
 FROM date, customer, supplier, part, lineorder
 WHERE  lo_custkey = c_custkey AND
        lo_suppkey = s_suppkey AND
        lo_partkey = p_partkey AND
        lo_orderdate = d_datekey );
```

VFV Q4.2:

```
CREATE TABLE vfv_q4_2 AS (
 SELECT d_year, s_nation, p_category, lo_revenue, lo_supplycost,
   c_region, s_region, p_mfgr
 FROM date, customer, supplier, part, lineorder
```

```
4  WHERE  lo_custkey = c_custkey AND
5         lo_suppkey = s_suppkey AND
6         lo_partkey = p_partkey AND
7         lo_orderdate = d_datekey );
```

VFV Q4.3:

```
1 CREATE TABLE vfv_q4_3 AS (
2  SELECT d_year, s_city, p_brand1, lo_revenue, lo_supplycost, c_region,
     s_nation, p_category
3  FROM date, customer, supplier, part, lineorder
4  WHERE  lo_custkey = c_custkey AND
5         lo_suppkey = s_suppkey AND
6         lo_partkey = p_partkey AND
7         lo_orderdate = d_datekey );
```

A seguir demonstro os comandos para criação das Visões Materializadas (VMs):
VM Q1.1:

```
1 CREATE TABLE vm_q1_1 AS (
2  SELECT d_year, lo_discount, lo_quantity, SUM (lo_extendedprice *
     lo_discount) as revenue
3     FROM lineorder, date
4     WHERE lo_orderdate = d_datekey
5     GROUP BY d_year, lo_discount, lo_quantity);
```

VM Q1.2:

```
1 CREATE TABLE vm_q1_2 AS (
2  SELECT SUM (lo_extendedprice * lo_discount) as revenue, d_yearmonthnum,
      lo_discount, lo_quantity
3     FROM lineorder, date
4     WHERE lo_orderdate = d_datekey
5     GROUP BY d_yearmonthnum, lo_discount, lo_quantity);
```

VM Q1.3:

```
1 CREATE TABLE vm_q1_3 AS (
2  SELECT SUM (lo_extendedprice*lo_discount) as revenue, d_weeknuminyear,
     d_year, lo_discount, lo_quantity
3     FROM lineorder, date
4     WHERE lo_orderdate = d_datekey
5     GROUP BY d_weeknuminyear, d_year, lo_discount, lo_quantity);
```

VM Q2.1:

```
1 CREATE TABLE vm_q2_1 AS (
2  SELECT sum(lo_revenue) as revenue, d_year, p_brand1, p_category,
     s_region
3     FROM lineorder, date, part, supplier
4     WHERE lo_orderdate = d_datekey AND
5     lo_partkey = p_partkey AND
```

```
6     lo_suppkey = s_suppkey
7     GROUP BY d_year, p_brand1, p_category, s_region );
```

VM Q2.2:

```
1 CREATE TABLE vm_q2_2 AS (
2  SELECT sum(lo_revenue) as revenue, d_year, p_brand1, s_region
3     FROM lineorder, date, part, supplier
4     WHERE lo_orderdate = d_datekey AND
5     lo_partkey = p_partkey AND
6     lo_suppkey = s_suppkey
7     GROUP BY d_year, p_brand1, s_region);
```

VM Q2.3:

```
1 CREATE TABLE vm_q2_3 AS (
2  SELECT sum(lo_revenue) as revenue, d_year, p_brand1, s_region
3     FROM lineorder, date, part, supplier
4     WHERE lo_orderdate = d_datekey AND
5     lo_partkey = p_partkey AND
6     lo_suppkey = s_suppkey
7     GROUP BY d_year, p_brand1, s_region );
```

VM Q3.1:

```
1 CREATE TABLE vm_q3_1 AS (
2  SELECT c_nation, s_nation, c_region, s_region, d_year, SUM(lo_revenue)
     AS revenue
3  FROM customer, lineorder, supplier, date
4  WHERE lo_custkey = c_custkey AND
5     lo_suppkey = s_suppkey AND
6     lo_orderdate = d_datekey
7  GROUP BY c_nation, s_nation, c_region, s_region, d_year );
```

VM Q3.2:

```
1 CREATE TABLE vm_q3_2 AS (
2  SELECT c_city, s_city, c_nation, s_nation, d_year, SUM(lo_revenue) AS
     revenue
3  FROM customer, lineorder, supplier, date
4  WHERE lo_custkey = c_custkey AND
5     lo_suppkey = s_suppkey AND
6     lo_orderdate = d_datekey
7  GROUP BY c_city, s_city, c_nation, s_nation, d_year );
```

VM Q3.3:

```
1 CREATE TABLE vm_q3_3 AS (
2  SELECT c_city, s_city, d_year, SUM(lo_revenue) AS revenue
3  FROM customer, lineorder, supplier, date
4  WHERE lo_custkey = c_custkey AND
5     lo_suppkey = s_suppkey AND
6     lo_orderdate = d_datekey
```

```
7  GROUP BY c_city, s_city, d_year );
```

VM Q3.4:

```
1 CREATE TABLE vm_q3_4 AS (
2  SELECT c_city, s_city, d_year, d_yearmonth, SUM(lo_revenue) AS revenue
3  FROM customer, lineorder, supplier, date
4  WHERE lo_custkey = c_custkey AND
5     lo_suppkey = s_suppkey AND
6     lo_orderdate = d_datekey
7  GROUP BY c_city, s_city, d_year, d_yearmonth );
```

VM Q4.1:

```
1 CREATE TABLE vm_q4_1 AS (
2  SELECT d_year, c_nation, c_region, s_region, p_mfgr, SUM(lo_revenue -
     lo_supplycost) AS profit
3  FROM date, customer, supplier, part, lineorder
4  WHERE lo_custkey = c_custkey AND
5      lo_suppkey = s_suppkey AND
6      lo_partkey = p_partkey AND
7      lo_orderdate = d_datekey
8  GROUP BY d_year, c_nation, c_region, s_region, p_mfgr );
```

VM Q4.2:

```
1 CREATE TABLE vm_q4_2 AS (
2  SELECT d_year, s_nation, p_category, c_region, s_region, p_mfgr, SUM(
     lo_revenue - lo_supplycost) AS profit
3  FROM date, customer, supplier, part, lineorder
4  WHERE lo_custkey = c_custkey AND
5      lo_suppkey = s_suppkey AND
6      lo_partkey = p_partkey AND
7      lo_orderdate = d_datekey
8  GROUP BY d_year, s_nation, p_category, c_region, s_region, p_mfgr );
```

VM Q4.3:

```
1 CREATE TABLE vm_q4_3 AS (
2  SELECT d_year, s_city, p_brand1, c_region, s_nation, p_category, SUM(
     lo_revenue - lo_supplycost) AS profit
3  FROM date, customer, supplier, part, lineorder
4  WHERE lo_custkey = c_custkey AND
5      lo_suppkey = s_suppkey AND
6      lo_partkey = p_partkey AND
7      lo_orderdate = d_datekey
8  GROUP BY d_year, s_city, p_brand1, c_region, s_nation, p_category );
```

Após a criação de toda a estrutura do banco de dados, é processada as consultas nas visões. Iniciaremos a demonstração das consultas na Junção Estrela, estrutura base do banco de dados escolhido. Posteriormente é apresentado as consultas nas VFVs e VMs.

Q1.1:

```
SELECT SUM(lo_extendedprice*lo_discount) AS revenue
 FROM lineorder, date
 WHERE lo_orderdate = d_datekey AND
     d_year = 1993 AND
     lo_discount BETWEEN 1 AND 3 AND
     lo_quantity < 25;
```

Q1.2:

```
SELECT SUM(lo_extendedprice*lo_discount) AS revenue
 FROM lineorder, date
 WHERE lo_orderdate = d_datekey AND
     d_yearmonthnum = 199401 AND
     lo_discount BETWEEN 4 AND 6 AND
     lo_quantity BETWEEN 26 AND 35;
```

Q1.3:

```
SELECT SUM(lo_extendedprice*lo_discount) AS revenue
 FROM lineorder, date
 WHERE lo_orderdate = d_datekey AND
     d_weeknuminyear = 6 AND
     d_year = 1994 AND
     lo_discount BETWEEN 5 AND 7 AND
     lo_quantity BETWEEN 26 AND 35;
```

Q2.1:

```
SELECT SUM(lo_revenue), d_year, p_brand1
 FROM lineorder, date, part, supplier
 WHERE lo_orderdate = d_datekey AND
    lo_partkey = p_partkey AND
    lo_suppkey = s_suppkey AND
    p_category = 'MFGR#12' AND
    s_region = 'AMERICA'
 GROUP BY d_year, p_brand1
 ORDER BY d_year, p_brand1;
```

Q2.2:

```
SELECT SUM(lo_revenue), d_year, p_brand1
 FROM lineorder, date, part, supplier
 WHERE lo_orderdate = d_datekey AND
     lo_partkey = p_partkey AND
     lo_suppkey = s_suppkey AND
     p_brand1 BETWEEN 'MFGR#2221' AND 'MFGR#2228' AND
     s_region = 'ASIA'
 GROUP BY d_year, p_brand1
 ORDER BY d_year, p_brand1;
```

Q2.3:

```
SELECT SUM(lo_revenue), d_year, p_brand1
```

```
 FROM lineorder, date, part, supplier
 WHERE lo_orderdate = d_datekey AND
     lo_partkey = p_partkey AND
     lo_suppkey = s_suppkey AND
     p_brand1 = 'MFGR#2221' AND
     s_region = 'EUROPE'
 GROUP BY d_year, p_brand1
 ORDER BY d_year, p_brand1;
```

Q3.1:

```
SELECT c_nation, s_nation, d_year, SUM(lo_revenue) AS revenue
 FROM customer, lineorder, supplier, date
 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;
```

Q3.2:

```
SELECT c_city, s_city, d_year, SUM(lo_revenue) AS revenue
 FROM customer, lineorder, supplier, date
 WHERE lo_custkey = c_custkey AND
     lo_suppkey = s_suppkey AND
     lo_orderdate = d_datekey AND
     c_nation = 'UNITED STATES' 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;
```

Q3.3:

```
SELECT c_city, s_city, d_year, SUM(lo_revenue) AS revenue
 FROM customer, lineorder, supplier, date
 WHERE 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 >= 1992 AND d_year <= 1997
 GROUP BY c_city, s_city, d_year
 ORDER BY d_year ASC, revenue DESC;
```

Q3.4:

```
SELECT c_city, s_city, d_year, SUM(lo_revenue) AS revenue
 FROM customer, lineorder, supplier, date
```

```
 WHERE 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_yearmonth = 'Dec1997'
 GROUP BY c_city, s_city, d_year
 ORDER BY d_year ASC, revenue DESC;
```

Q4.1:

```
SELECT d_year, c_nation, SUM(lo_revenue - lo_supplycost) AS profit
 FROM date, 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;
```

Q4.2:

```
SELECT d_year, s_nation, p_category, SUM(lo_revenue - lo_supplycost) AS
   profit
 FROM date, 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;
```

Q4.3:

```
SELECT d_year, s_city, p_brand1, SUM(lo_revenue - lo_supplycost) AS
   profit
 FROM date, 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
```

```
10     p_category = 'MFGR#14'
11  GROUP BY d_year, s_city, p_brand1
12  ORDER BY d_year, s_city, p_brand1;
```

A seguir é exibido as consultas utilizadas neste trabalho para as VFVs:

VFV Q1.1:

```
1 SELECT SUM(lo_extendedprice*lo_discount) AS revenue
2  FROM vfv_q1_1
3  WHERE d_year = 1993 AND
4      lo_discount BETWEEN 1 AND 3 AND
5      lo_quantity < 25;
```

VFV Q1.2:

```
1 SELECT SUM(lo_extendedprice*lo_discount) AS revenue
2  FROM vfv_q1_2
3  WHERE d_yearmonthnum = 199401 AND
4      lo_discount BETWEEN 4 AND 6 AND
5      lo_quantity BETWEEN 26 AND 35;
```

VFV Q1.3:

```
1 SELECT SUM(lo_extendedprice*lo_discount) AS revenue
2  FROM vfv_q1_3
3  WHERE d_weeknuminyear = 6 AND
4      d_year = 1994 AND
5      lo_discount BETWEEN 5 AND 7 AND
6      lo_quantity BETWEEN 26 AND 35;
```

VFV Q2.1:

```
1 SELECT SUM(lo_revenue), d_year, p_brand1
2  FROM vfv_q2_1
3  WHERE p_category = 'MFGR#12' AND
4     s_region = 'AMERICA'
5  GROUP BY d_year, p_brand1
6  ORDER BY d_year, p_brand1;
```

VFV Q2.2:

```
1 SELECT SUM(lo_revenue), d_year, p_brand1
2  FROM vfv_q2_2
3  WHERE p_brand1 BETWEEN 'MFGR#2221' AND 'MFGR#2228' AND
4      s_region = 'ASIA'
5  GROUP BY d_year, p_brand1
6  ORDER BY d_year, p_brand1;
```

VFV Q2.3:

```
1 SELECT SUM(lo_revenue), d_year, p_brand1
2  FROM vfv_q2_3
3  WHERE p_brand1 = 'MFGR#2221' AND
```

```
4      s_region = 'EUROPE'
5  GROUP BY d_year, p_brand1
6  ORDER BY d_year, p_brand1;
```

VFV Q3.1:

```
1 SELECT c_nation, s_nation, d_year, SUM(lo_revenue) AS revenue
2  FROM vfv_q3_1
3  WHERE c_region = 'ASIA' AND
4      s_region = 'ASIA' AND
5      d_year >= 1992 AND
6      d_year <= 1997
7  GROUP BY c_nation, s_nation, d_year
8  ORDER BY d_year ASC, revenue DESC;
```

VFV Q3.2:

```
1 SELECT c_city, s_city, d_year, SUM(lo_revenue) AS revenue
2  FROM vfv_q3_2
3  WHERE c_nation = 'UNITED STATES' AND
4      s_nation = 'UNITED STATES' AND
5      d_year >= 1992 AND d_year <= 1997
6  GROUP BY c_city, s_city, d_year
7  ORDER BY d_year ASC, revenue DESC;
```

VFV Q3.3:

```
1 SELECT c_city, s_city, d_year, SUM(lo_revenue) AS revenue
2  FROM vfv_q3_3
3  WHERE (c_city='UNITED KI1'  OR c_city='UNITED KI5') AND
4     (s_city='UNITED KI1'  OR s_city='UNITED KI5') AND
5     d_year >= 1992 AND d_year <= 1997
6  GROUP BY c_city, s_city, d_year
7  ORDER BY d_year ASC, revenue DESC;
```

VFV Q3.4:

```
1 SELECT c_city, s_city, d_year, SUM(lo_revenue) AS revenue
2  FROM vfv_q3_4
3  WHERE (c_city='UNITED KI1' OR c_city='UNITED KI5') AND
4      (s_city='UNITED KI1' OR  s_city='UNITED KI5') AND
5      d_yearmonth = 'Dec1997'
6  GROUP BY c_city, s_city, d_year
7  ORDER BY d_year ASC, revenue DESC;
```

VFV Q4.1:

```
1 SELECT d_year, c_nation, SUM(lo_revenue - lo_supplycost) AS profit
2  FROM vfv_q4_1
3  WHERE c_region = 'AMERICA' AND
4      s_region = 'AMERICA' AND
5      (p_mfgr = 'MFGR#1' OR p_mfgr = 'MFGR#2')
6  GROUP BY d_year, c_nation
```

```
7 ORDER BY d_year, c_nation;
```

VFV Q4.2:

```
1 SELECT d_year, s_nation, p_category, SUM(lo_revenue - lo_supplycost) AS
    profit
2 FROM vfv_q4_2
3 WHERE c_region = 'AMERICA' AND
4    s_region = 'AMERICA' AND
5    (d_year = 1997 OR d_year = 1998) AND
6    (p_mfgr = 'MFGR#1' OR p_mfgr = 'MFGR#2')
7 GROUP BY d_year, s_nation, p_category
8 ORDER BY d_year, s_nation, p_category;
```

VFV Q4.3:

```
1 SELECT d_year, s_city, p_brand1, SUM(lo_revenue - lo_supplycost) AS
    profit
2 FROM vfv_q4_3
3 WHERE c_region = 'AMERICA' AND
4    s_nation = 'UNITED STATES' AND
5    (d_year = 1997 OR d_year = 1998) AND
6    p_category = 'MFGR#14'
7 GROUP BY d_year, s_city, p_brand1
8 ORDER BY d_year, s_city, p_brand1;
```

Para finalizar os comandos utilizados para o banco de dados PostgreSQL são exibidos a seguir as consultas para as VMs:

VM Q1.1:

```
1 SELECT SUM(revenue)
2 FROM vm_q1_1
3 WHERE d_year = 1993 AND
4     lo_discount BETWEEN 1 AND 3 AND
5     lo_quantity < 25;
```

VM Q1.2:

```
1 SELECT SUM(revenue)
2 FROM vm_q1_2
3 WHERE d_yearmonthnum = 199401 AND
4     lo_discount BETWEEN 4 AND 6 AND
5     lo_quantity BETWEEN 26 AND 35;
```

VM Q1.3:

```
1 SELECT SUM(revenue)
2 FROM vm_q1_3
3 WHERE d_weeknuminyear = 6 AND
4     d_year = 1994 AND
5     lo_discount BETWEEN 5 AND 7 AND
6     lo_quantity BETWEEN 26 AND 35;
```

VM Q2.1:

```
SELECT revenue, d_year, p_brand1
 FROM vm_q2_1
 WHERE p_category = 'MFGR#12' AND
    s_region = 'AMERICA'
 ORDER BY d_year, p_brand1;
```

VM Q2.2:

```
SELECT revenue, d_year, p_brand1
 FROM vm_q2_2
 WHERE p_brand1 BETWEEN 'MFGR#2221' AND 'MFGR#2228' AND
     s_region = 'ASIA'
 ORDER BY d_year, p_brand1;
```

VM Q2.3:

```
SELECT revenue, d_year, p_brand1
 FROM vm_q2_3
 WHERE p_brand1 = 'MFGR#2221' AND
     s_region = 'EUROPE'
 ORDER BY d_year, p_brand1;
```

VM Q3.1:

```
SELECT c_nation, s_nation, d_year, revenue
 FROM vm_q3_1
 WHERE c_region = 'ASIA' AND
     s_region = 'ASIA' AND
     d_year >= 1992 AND
     d_year <= 1997
 ORDER BY d_year ASC, revenue DESC;
```

VM Q3.2:

```
SELECT c_city, s_city, d_year, revenue
 FROM vm_q3_2
 WHERE c_nation = 'UNITED STATES' AND
     s_nation = 'UNITED STATES' AND
     d_year >= 1992 AND d_year <= 1997
 ORDER BY d_year ASC, revenue DESC;
```

VM Q3.3:

```
SELECT c_city, s_city, d_year, revenue
 FROM vm_q3_3
 WHERE (c_city='UNITED KI1'  OR c_city='UNITED KI5') AND
    (s_city='UNITED KI1'  OR s_city='UNITED KI5') AND
    d_year >= 1992 AND d_year <= 1997
 ORDER BY d_year ASC, revenue DESC;
```

VM Q3.4:

```
SELECT c_city, s_city, d_year, revenue
 FROM vm_q3_4
 WHERE (c_city='UNITED KI1' OR c_city='UNITED KI5') AND
     (s_city='UNITED KI1' OR  s_city='UNITED KI5') AND
     d_yearmonth = 'Dec1997'
 ORDER BY d_year ASC, revenue DESC;
```

VM Q4.1:

```
SELECT d_year, c_nation, profit
 FROM vm_q4_1
 WHERE c_region = 'AMERICA' AND
     s_region = 'AMERICA' AND
     (p_mfgr = 'MFGR#1' OR p_mfgr = 'MFGR#2')
 ORDER BY d_year, c_nation;
```

VM Q4.2:

```
SELECT d_year, s_nation, p_category, profit
 FROM vm_q4_2
 WHERE 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')
 ORDER BY d_year, s_nation, p_category;
```

VM Q4.3:

```
SELECT d_year, s_city, p_brand1, profit
 FROM vm_q4_3
 WHERE c_region = 'AMERICA' AND
    s_nation = 'UNITED STATES' AND
    (d_year = 1997 OR d_year = 1998) AND
    p_category = 'MFGR#14'
 ORDER BY d_year, s_city, p_brand1;
```

APÊNDICE B - Comandos no MongoDB

Neste apêndice é abordado todos os comandos realizados para o MongoDB. Sendo eles para criação das visões, referindo-se a visão fragmentada ou materializada, e também suas respectivas consultas conforme comentado durante o trabalho.

Em seguida é mostrado todos os comandos para a criação das Visões Fragmentadas (VFVs) para o MongoDB:

VFV Q1.1:

```
db.[COLLECTION].aggregate([{
 $project: {
  lo_extendedprice: 1,
  lo_discount: 1,
  lo_quantity: 1,
  'date.d_year': 1
 }
}, {
 $merge: {
  into: 'vfv_q1_1',
  whenMatched: 'replace',
  whenNotMatched: 'insert'
 }
}]
```

VFV Q1.2:

```
db.[COLLECTION].aggregate([{
 $project: {
  lo_extendedprice: 1,
  lo_discount: 1,
  lo_quantity: 1,
  'date.d_yearmonthnum': 1
 }
}, {
 $merge: {
  into: 'vfv_q1_2',
  whenMatched: 'replace',
  whenNotMatched: 'insert'
 }
}]
```

VFV Q1.3:

```
db.[COLLECTION].aggregate([{
  $project: {
  lo_extendedprice: 1,
  lo_discount: 1,
  lo_quantity: 1,
  'date.d_weeknuminyear': 1,
  'date.d_year': 1
 }
```

```
}, {
 $merge: {
  into: 'vfv_q1_3',
  whenMatched: 'replace',
  whenNotMatched: 'insert'
 }
}]
```

VFV Q2.1:

```
db.[COLLECTION].aggregate([{
  $project: {
  lo_revenue: 1,
  'date.d_year': 1,
  'part.p_brand1': 1,
  'part.p_category': 1,
  'supplier.s_region': 1
 }
}, {
 $merge: {
  into: 'vfv_q2_1',
  whenMatched: 'replace',
  whenNotMatched: 'insert'
 }
}]
```

VFV Q2.2:

```
db.[COLLECTION].aggregate([{
 $project: {
  lo_revenue: 1,
  'date.d_year': 1,
  'part.p_brand1': 1,
  'supplier.s_region': 1
 }
}, {
 $merge: {
  into: 'vfv_q2_1',
  whenMatched: 'replace',
  whenNotMatched: 'insert'
 }
}]
```

VFV Q2.3:

```
db.[COLLECTION].aggregate([{
 $project: {
  lo_revenue: 1,
  'date.d_year': 1,
  'part.p_brand1': 1,
  'supplier.s_region': 1
```

```
 }
}, {
 $merge: {
  into: 'vfv_q2_1',
  whenMatched: 'replace',
  whenNotMatched: 'insert'
 }
}]
```

VFV Q3.1:

```
db.[COLLECTION].aggregate([{
 $project: {
  'customer.c_nation': 1,
  'supplier.s_nation': 1,
  'customer.c_region': 1,
  'supplier.s_region': 1,
  'date.d_year': 1,
  lo_revenue: 1
 }
}, {
 $merge: {
  into: 'vfv_q3_1',
  whenMatched: 'replace',
  whenNotMatched: 'insert'
 }
}]
```

VFV Q3.2:

```
db.[COLLECTION].aggregate([{
 $project: {
  'customer.c_city': 1,
  'supplier.s_city': 1,
  'customer.c_nation': 1,
  'supplier.s_nation': 1,
  'date.d_year': 1,
  lo_revenue: 1
 }
}, {
 $merge: {
  into: 'vfv_q3_2',
  whenMatched: 'replace',
  whenNotMatched: 'insert'
 }
}]
```

VFV Q3.3:

```
db.[COLLECTION].aggregate([{
 $project: {
```

```
  'customer.c_city': 1,
  'supplier.s_city': 1,
  'date.d_year': 1,
  lo_revenue: 1
 }
}, {
 $merge: {
  into: 'vfv_q3_3',
  whenMatched: 'replace',
  whenNotMatched: 'insert'
 }
}]
```

VFV Q3.4:

```
db.[COLLECTION].aggregate([{
 $project: {
  'customer.c_city': 1,
  'supplier.s_city': 1,
  'date.d_year': 1,
  'date.d_yearmonth': 1,
  lo_revenue: 1
 }
}, {
 $merge: {
  into: 'vfv_q3_4',
  whenMatched: 'replace',
  whenNotMatched: 'insert'
 }
}]
```

VFV Q4.1:

```
db.[COLLECTION].aggregate([{
 $project: {
  'date.d_year': 1,
  'customer.c_nation': 1,
  lo_revenue: 1,
  lo_supplycost: 1,
  'customer.c_region': 1,
  'supplier.s_region': 1,
  'part.p_mfgr': 1
 }
}, {
 $merge: {
  into: 'vfv_q4_1',
  whenMatched: 'replace',
  whenNotMatched: 'insert'
 }
}]
```

VFV Q4.2:

```
db.[COLLECTION].aggregate([{
 $project: {
  'date.d_year': 1,
  'supplier.s_nation': 1,
  'part.p_category': 1,
  lo_revenue: 1,
  lo_supplycost: 1,
  'customer.c_region': 1,
  'supplier.s_region': 1,
  'part.p_mfgr': 1
 }
}, {
 $merge: {
  into: 'vfv_q4_2',
  whenMatched: 'replace',
  whenNotMatched: 'insert'
 }
}]
```

VFV Q4.3:

```
db.[COLLECTION].aggregate([{
 $project: {
  'date.d_year': 1,
  'supplier.s_city': 1,
  'part.p_brand1': 1,
  lo_revenue: 1,
  lo_supplycost: 1,
  'customer.c_region': 1,
  'supplier.s_nation': 1,
  'part.p_category': 1
 }
}, {
 $merge: {
  into: 'vfv_q4_3',
  whenMatched: 'replace',
  whenNotMatched: 'insert'
 }
}]
```

A seguir demonstro os comandos para criação das Visões Materializadas (VMs):
VM Q1.1:

```
db.[COLLECTION].aggregate([{
 $project:{
   'date.d_year': 1,
   'lo_discount': 1,
   'lo_quantity': 1,
```

```
    'lo_extendedprice': 1,
    'lo_revenue': 1,
    'customer.c_city': 1
   }
  },{
  $group:{
    '_id':{
      'd_year': '$date.d_year',
      'lo_discount': '$lo_discount',
      'lo_quantity': '$lo_quantity'
    },
    'revenue':{ '$sum': '$lo_revenue'}
    }
   },{
  $merge:{
    'into': 'vm_q1_1',
    'whenMatched': 'replace',
    'whenNotMatched': 'insert'
   }
}]
```

VM Q1.2:

```
db.[COLLECTION].aggregate([{
 $project: {
  lo_extendedprice: 1,
  lo_discount: 1,
  'date.d_yearmonthnum': 1,
  lo_quantity: 1
 }
}, {
 $group: {
  _id: {
   lo_extendedprice: '$lo_extendedprice',
   lo_discount: '$lo_discount',
   d_yearmonthnum: '$date.d_yearmonthnum',
   lo_quantity: '$lo_quantity'
  },
  revenue: {
   $sum: {
    $multiply: [
     '$lo_extendedprice',
     '$lo_quantity'
    ]
   }
  }
 }
}, {
```

```
 $merge: {
  into: 'vm_q1_2',
  whenMatched: 'replace',
  whenNotMatched: 'insert'
 }
}]
```

VM Q1.3:

```
db.[COLLECTION].aggregate([{
  $project: {
   'lo_extendedprice': 1,
   'lo_discount': 1,
   'date.d_weeknuminyear': 1,
   'date.d_year': 1,
   'lo_quantity': 1
  }
  },{
  $group: {
   '_id': {
     'lo_extendedprice': '$lo_extendedprice',
     'lo_discount': '$lo_discount',
     'd_weeknuminyear': '$date.d_weeknuminyear',
     'd_year': '$date.d_year',
     'lo_quantity': '$lo_quantity'
   },
   'revenue': {
      '$sum': { '$multiply': [ '$lo_extendedprice', '$lo_quantity']}
     }
  }
  },{
  $merge: {
   'into': 'vm_q1_3',
   'whenMatched': 'replace',
   'whenNotMatched': 'insert'
  }
}]
```

VM Q2.1:

```
db.[COLLECTION].aggregate([{

}]
```

VM Q2.2:

```
db.[COLLECTION].aggregate([{
  $project: {
   lo_revenue: 1,
   'date.d_year': 1,
   'part.p_brand1': 1,
```

```
    'part.p_category': 1,
    'supplier.s_region': 1
  }
}, {
  $group: {
    _id: {
      d_year: '$date.d_year',
      p_brand1: '$part.p_brand1',
      p_category: '$part.p_category',
      s_region: '$supplier.s_region'
    },
    revenue: {
      $sum: '$lo_revenue'
    }
  }
}, {
  $merge: {
    into: 'vm_q2_2',
    whenMatched: 'replace',
    whenNotMatched: 'insert'
  }
}]
```

VM Q2.3:

```
db.[COLLECTION].aggregate([{
  $project: {
    lo_revenue: 1,
    'date.d_year': 1,
    'part.p_brand1': 1,
    'part.p_category': 1,
    'supplier.s_region': 1
  }
}, {
  $group: {
    _id: {
      d_year: '$date.d_year',
      p_brand1: '$part.p_brand1',
      p_category: '$part.p_category',
      s_region: '$supplier.s_region'
    },
    revenue: {
      $sum: '$lo_revenue'
    }
  }
}, {
  $merge: {
    into: 'vm_q2_3',
```

```
  whenMatched: 'replace',
  whenNotMatched: 'insert'
 }
}]
```

VM Q3.1:

```
db.[COLLECTION].aggregate([{
  $project: {
   lo_revenue: 1,
   'date.d_year': 1,
   'supplier.s_region': 1,
   'supplier.s_nation': 1,
   'customer.c_nation': 1,
   'customer.c_region': 1
 }
}, {
 $group: {
  _id: {
   d_year: '$date.d_year',
   s_region: '$supplier.s_region',
   c_region: '$customer.c_region',
   s_nation: '$supplier.s_nation',
   c_nation: '$customer.c_nation'
  },
  revenue: {
   $sum: '$lo_revenue'
  }
 }
}, {
 $merge: {
  into: 'vm_q3_1',
  whenMatched: 'replace',
  whenNotMatched: 'insert'
 }
}]
```

VM Q3.2:

```
db.[COLLECTION].aggregate([{
  $project: {
   lo_revenue: 1,
   'date.d_year': 1,
   'supplier.s_city': 1,
   'supplier.s_nation': 1,
   'customer.c_nation': 1,
   'customer.c_city': 1
 }
}, {
 $group: {
```

```
  _id: {
   d_year: '$date.d_year',
   s_city: '$supplier.s_city',
   c_city: '$customer.c_city',
   s_nation: '$supplier.s_nation',
   c_nation: '$customer.c_nation'
  },
  revenue: {
   $sum: '$lo_revenue'
  }
 }
}, {
 $merge: {
  into: 'vm_q3_2',
  whenMatched: 'replace',
  whenNotMatched: 'insert'
 }
}]
```

VM Q3.3:

```
db.[COLLECTION].aggregate([{
  $project: {
   lo_revenue: 1,
   'date.d_year': 1,
   'supplier.s_city': 1,
   'customer.c_city': 1
 }
}, {
 $group: {
  _id: {
   d_year: '$date.d_year',
   s_city: '$supplier.s_city',
   c_city: '$customer.c_city'
  },
  revenue: {
   $sum: '$lo_revenue'
  }
 }
}, {
 $merge: {
   into: 'vm_q3_3',
   whenMatched: 'replace',
   whenNotMatched: 'insert'
 }
}]
```

VM Q3.4:

```
db.[COLLECTION].aggregate([{
```

```
  $project: {
   lo_revenue: 1,
   'date.d_year': 1,
   'date.d_yearmonth': 1,
   'supplier.s_city': 1,
   'customer.c_city': 1
 }
}, {
 $group: {
  _id: {
   d_year: '$date.d_year',
   d_yearmonth: '$date.d_yearmonth',
   s_city: '$supplier.s_city',
   c_city: '$customer.c_city'
  },
  revenue: {
   $sum: '$lo_revenue'
   }
  }
 },{
 $merge: {
   into: 'vm_q3_4',
   whenMatched: 'replace',
   whenNotMatched: 'insert'
 }
}]
```

VM Q4.1:

```
db.[COLLECTION].aggregate([{
  $project: {
   lo_revenue: 1,
   lo_supplycost: 1,
   'date.d_year': 1,
   'supplier.s_region': 1,
   'customer.c_nation': 1,
   'customer.c_region': 1,
   'part.p_mfgr': 1
  }
 },{
 $group: {
  _id: {
   d_year: '$date.d_year',
   c_nation: '$customer.c_nation'
  },
  profit: {
   $sum: {
    $subtract: ['$lo_revenue','$lo_supplycost']
```

```
    }
   }
  }
 },{
 $merge: {
   into: 'vm_q4_1',
   whenMatched: 'replace',
   whenNotMatched: 'insert'
 }
}]
```

VM Q4.2:

```
db.[COLLECTION].aggregate([{
  $project: {
    'lo_revenue': 1,
    'lo_supplycost': 1,
    'date.d_year': 1,
    'supplier.s_city': 1,
    'supplier.s_nation': 1,
    'customer.c_nation': 1,
    'customer.c_region': 1,
    'part.p_brand1': 1,
    'part.p_category': 1
        }
  }, {
  $group: {
    '_id': {
      'd_year': '$date.d_year',
      's_city': '$supplier.s_city',
      'p_brand1': '$part.p_brand1'
    },
    'profit': {
      '$sum': {
        '$subtract': ['$lo_revenue', '$lo_supplycost']
        }
    }
  }
  }, {
  $merge: {
    'into': 'vm_q4_3',
    'whenMatched': 'replace',
    'whenNotMatched': 'insert'}
  }
}]
```

VM Q4.3:

```
db.[COLLECTION].aggregate([{

```

```
}]
```

Após a criação de toda a estrutura do banco de dados, é processada as consultas nas visões. Iniciaremos a demonstração das consultas na Junção Estrela. Posteriormente é apresentado as consultas nas VFVs e VMs.

Q1.1:

```
db.[COLLECTION].aggregate([{
 $match: {
  'date.d_year': {$eq: 1993},
  lo_quantity: {$lt: 25},
  lo_discount: {$gte: 1, $lte: 3}
 }
},{
 $group: {
  _id: null,
  revenue: {
    $sum: { $multiply: ['$lo_extendedprice', '$lo_discount']}
  }
 }
}]
```

Q1.2:

```
db.[COLLECTION].aggregate([{
 $match: {
  'date.d_yearmonthnum': {$eq: 199401},
  lo_discount: {$gte: 4, $lte: 6},
  lo_quantity: {$gte: 26, $lte: 35}
 }
 },{
 $group: {
  _id: null,
  revenue: {
   $sum: { $multiply: ['$lo_extendedprice', '$lo_discount']}
  }
 }
}]
```

Q1.3:

```
db.[COLLECTION].aggregate([{
 $match: {
  'date.d_weeknuminyear': {$eq: 6},
  'date.d_year': {$eq: 1994},
  lo_discount: {$gte: 5, $lte: 7},
  lo_quantity: {$gte: 26, $lte: 35}
 }
 },{
 $group: {
```

```
  _id: null,
  revenue: {
   $sum: {$multiply: ['$lo_extendedprice', '$lo_discount']}
  }
 }
}]
```

Q2.1:

```
db.[COLLECTION].aggregate([{
 $match: {
   'part.p_category': 'MFGR#12',
   'supplier.0.s_region': {$eq: 'AMERICA'}
  }
 }, {
 $group: {
   _id:{ d_year: '$date.d_year',
         p_brand1: '$part.p_brand1'
   },
  revenue: {
         $sum: '$lo_revenue'
  }
 }
}]
```

Q2.2:

```
db.[COLLECTION].aggregate([{
 $match: {
   'part.p_brand1': {$in: ['MFGR#2221', 'MFGR#2228']},
   'supplier.s_region': {$eq: 'ASIA'}
 }
 },{
 $group: {
   _id: { d_year: '$date.d_year',
          p_brand1: '$part.p_brand1'
   },
   revenue: {
     $sum: '$lo_revenue'
   }
 }
}]
```

Q2.3:

```
db.[COLLECTION].aggregate([{
 $match: {
  'part.p_brand1': {$eq: 'MFGR#2221'},
  'supplier.s_region': {$eq: 'EUROPE'}
  }
 },{
```

```
 $group: {
   _id: { d_year: '$date.d_year',
          p_brand1: '$part.p_brand1'
   },
   revenue: {$sum: '$lo_revenue'}
 }
}]
```

Q3.1:

```
db.[COLLECTION].aggregate([{
 $match: {
  'date.d_year': {$in: [1992, 1997]},
  'supplier.s_region': {$eq: 'ASIA'},
  'customer.c_region': 'ASIA'
 }
}, {
 $group: {
  _id: {
   c_nation: '$customer.c_nation',
   s_nation: '$supplier.s_nation',
   d_year: '$date.d_year'
  },
  revenue: {$sum: '$lo_revenue'}
 }
}, {
 $sort: {
  d_year: 1,
  revenue: -1
 }
}]
```

Q3.2:

```
db.[COLLECTION].aggregate([{
 $match: {
  'date.d_year': {$in: [1992, 1997]}
 }
}, {
 $match: {
  $or: [{'customer.c_city': {$eq: 'UNITED KI1'}},
        {'customer.c_city': {$eq: 'UNITED KI5'}}]
 }
}, {
 $match: {
  $or: [{'supplier.s_city': {$eq: 'UNITED KI1'}},
        {'supplier.s_city': {$eq: 'UNITED KI5'}}]
 }
}, {
 $group: {
```

```
  _id: {
   c_city: '$customer.c_city',
   s_city: '$supplier.s_city',
   d_year: '$date.d_year'
  },
  revenue: {$sum: '$lo_revenue'}
 }
}, {
 $sort: {
  d_year: 1,
  revenue: -1
 }
}]
```

Q3.3:

```
db.[COLLECTION].aggregate([{
 $match: {
  'date.d_year': {$in: [1992, 1997]}
 }
}, {
 $match: {
  $or: [{'customer.c_city': {$eq: 'UNITED KI1'}},
        {'customer.c_city': {$eq: 'UNITED KI5'}}]
 }
}, {
 $match: {
  $or: [{'supplier.s_city': {$eq: 'UNITED KI1'}},
        {'supplier.s_city': {$eq: 'UNITED KI5'}}]
 }
}, {
 $group: {
  _id: {
   c_city: '$customer.c_city',
   s_city: '$supplier.s_city',
   d_year: '$date.d_year'
  },
  revenue: {$sum: '$lo_revenue'}
 }
}, {
 $sort: {
  d_year: 1,
  revenue: -1
 }
}]
```

Q3.4:

```
db.[COLLECTION].aggregate([{
 $match: {
```

```
  'date.d_yearmonth': 'Dec1997'
 }
}, {
 $match: {
  $or: [{'customer.c_city': {$eq: 'UNITED KI1'}},
        {'customer.c_city': {$eq: 'UNITED KI5'}}]
 }
}, {
 $match: {
  $or: [{'supplier.s_city': {$eq: 'UNITED KI1'}},
        {'supplier.s_city': {$eq: 'UNITED KI5'}}]
 }
}, {
 $group: {
  _id: {
   c_city: '$customer.c_city',
   s_city: '$supplier.s_city',
   d_year: '$date.d_year'
  },
  revenue: {$sum: '$lo_revenue'}
 }
}, {
 $sort: {
  d_year: 1,
  revenue: -1
 }
}]
```

Q4.1:

```
db.[COLLECTION].aggregate([{
 $match: {
  'customer.c_region': {$eq: 'AMERICA'},
  'supplier.s_region': {$eq: 'AMERICA'}
 }
}, {
 $match: {
  $or: [{'part.p_mfgr': {$eq: 'MFGR#1'}},
        {'part.p_mfgr': {$eq: 'MFGR#2'}}]
 }
}, {
 $group: {
  _id: {
   d_year: '$date.d_year',
   c_nation: '$customer.c_nation'
  },
  profit: {
   $sum: { $subtract: ['$lo_revenue', '$lo_supplycost']}
```

```
  }
 }
}]
```

Q4.2:

```
db.[COLLECTION].aggregate([{
 $match: {
  'customer.c_region': { $eq: 'AMERICA'},
  'supplier.s_region': { $eq: 'AMERICA'}
 }
}, {
 $match: {
  $or: [{'date.d_year': {$eq: 1997}},
        {'date.d_year': {$eq: 1998}}]
 }
}, {
 $match: {
  $or: [{'part.p_mfgr': {$eq: 'MFGR#1'}},
        {'part.p_mfgr': {$eq: 'MFGR#2'}}]
 }
}, {
 $group: {
  _id: {
   d_year: '$date.d_year',
   s_nation: '$supplier.s_nation',
   p_category: '$part.p_category'
  },
  profit: {
   $sum: {$subtract: ['$lo_revenue', '$lo_supplycost']}
  }
 }
}, {
 $sort: {
  d_year: 1,
  s_nation: 1,
  p_category: 1
 }
}]
```

Q4.3:

```
db.[COLLECTION].aggregate([{
 $match: {
  'customer.c_region': {$eq: 'AMERICA'},
  'supplier.s_nation': {$eq: 'UNITED STATES'},
  'part.p_category': {$eq: 'MFGR#14'}
 }
}, {
 $match: {
```

```
  $or: [{'date.d_year': {$eq: 1997}},
        {'date.d_year': {$eq: 1998}}]
 }
}, {
 $group: {
  _id: {
   d_year: '$date.d_year',
   s_city: '$supplier.s_city',
   p_brand1: '$part.p_brand1'
  },
  profit: {
   $sum: { $subtract: ['$lo_revenue', '$lo_supplycost']}
  }
 }
}, {
 $sort: {
  d_year: 1,
  s_city: 1,
  p_brand1: 1
 }
}]
```

A seguir é exibido as consultas utilizadas neste trabalho para as VFVs:

VFV Q1.1:

```
db.[COLLECTION].aggregate([{
 $match: {
  'date.d_year': {$eq: 1993},
  lo_quantity: {$lt: 25},
  lo_discount: {$gte: 1, $lte: 3}
 }
}, {
 $group: {
  _id: null,
  revenue: {
   $sum: {$multiply: ['$lo_extendedprice','$lo_discount']}
  }
 }
}]
```

VFV Q1.2:

```
db.[COLLECTION].aggregate([{
 $match: {
  'date.d_yearmonthnum': {$eq: 199401},
  lo_discount: {$gte: 4, $lte: 6},
  lo_quantity: {$gte: 26, $lte: 35}
 }
}, {
 $group: {
```

```
  _id: null,
  revenue: {
   $sum: {$multiply: ['$lo_extendedprice', '$lo_discount']}
  }
 }
}]
```

VFV Q1.3:

```
db.[COLLECTION].aggregate([{
 $match: {
  'date.d_weeknuminyear': {$eq: 6},
  'date.d_year': {$eq: 1994},
  lo_discount: {$gte: 5, $lte: 7},
  lo_quantity: {$gte: 26, $lte: 35}
 }
}, {
 $group: {
  _id: null,
  revenue: {
   $sum: {$multiply: ['$lo_extendedprice', '$lo_discount']}
  }
 }
}]
```

VFV Q2.1:

```
db.[COLLECTION].aggregate([{
 $match: {
  'part.0.p_category': 'MFGR#12',
  'supplier.0.s_region': {$eq: 'AMERICA'}
 }
}, {
 $group: {
  _id: {
   d_year: '$date.d_year',
   p_brand1: '$part.p_brand1'
  },
  revenue: {$sum: '$lo_revenue'}
 }
}]
```

VFV Q2.2:

```
db.[COLLECTION].aggregate([{
 $match: {
  'part.p_brand1': {$in: ['MFGR#2221', 'MFGR#2228']},
  'supplier.s_region': {$eq: 'ASIA'}
 }
}, {
 $group: {
```

```
  _id: {
   d_year: '$date.d_year',
   p_brand1: '$part.p_brand1'
  },
  revenue: {$sum: '$lo_revenue'}
 }
}, {
 $sort: {
  'date.d_year': 1,
  'part.p_brand1': 1,
  _id: 1
 }
}]
```

VFV Q2.3:

```
db.[COLLECTION].aggregate([{
 $match: {
  'part.p_brand1': {$eq: 'MFGR#2221'},
  'supplier.s_region': {$eq: 'EUROPE'}
 }
}, {
 $group: {
  _id: {
   d_year: '$date.d_year',
   p_brand1: '$part.p_brand1'
  },
  revenue: {$sum: '$lo_revenue'}
 }
}, {
 $sort: {
  'date.d_year': 1,
  'part.p_brand1': 1,
  _id: 1
 }
}]
```

VFV Q3.1:

```
db.[COLLECTION].aggregate([{
 $match: {
  'date.d_year': {$in: [1992, 1997]},
  'supplier.s_region': {$eq: 'ASIA'},
  'customer.c_region': {$eq: 'ASIA'}
 }
}, {
 $group: {
  _id: {
   c_nation: '$customer.c_nation',
   s_nation: '$supplier.s_nation',
```

```
    d_year: '$date.d_year'
   },
   revenue: {$sum: '$lo_revenue'}
  }
}, {
  $sort: {
   d_year: 1,
   revenue: -1,
   _id: 1
  }
}]
```

VFV Q3.2:

```
db.[COLLECTION].aggregate([{
  $match: {
   'date.d_year': {$in: [1992, 1997]},
   'supplier.s_nation': 'UNITED STATES',
   'customer.c_nation': 'UNITED STATES'
  }
}, {
  $group: {
   _id: {
    c_city: '$customer.c_city',
    s_city: '$supplier.s_city',
    d_year: '$date.d_year'
   },
   revenue: {$sum: '$lo_revenue'}
  }
}]
```

VFV Q3.3:

```
db.[COLLECTION].aggregate([{
  $match: {
   'date.d_year': {$in: [1992, 1997]}
  }
}, {
  $match: {
   $or: [{'customer.c_city': {$eq: 'UNITED KI1'}},
         {'customer.c_city': {$eq: 'UNITED KI5'}}]
  }
}, {
  $match: {
   $or: [{'supplier.s_city': {$eq: 'UNITED KI1'}},
         {'supplier.s_city': {$eq: 'UNITED KI5'}}]
  }
}, {
  $group: {
   _id: {
```

```
    c_city: '$customer.c_city',
    s_city: '$supplier.s_city',
    d_year: '$date.d_year'
   },
   revenue: {$sum: '$lo_revenue'}
 }
}]
```

VFV Q3.4:

```
db.[COLLECTION].aggregate([{
 $match: {
  'date.d_yearmonth': 'Dec1997'
 }
}, {
 $match: {
  $or: [{'customer.c_city': {$eq: 'UNITED KI1'}},
        {'customer.c_city': {$eq: 'UNITED KI5'}}]
 }
}, {
 $match: {
  $or: [{'supplier.s_city': {$eq: 'UNITED KI1'}},
        {'supplier.s_city': {$eq: 'UNITED KI5'}}]
 }
}, {
 $group: {
  _id: {
   c_city: '$customer.c_city',
   s_city: '$supplier.s_city',
   d_year: '$date.d_year'
  },
  revenue: {$sum: '$lo_revenue'}
 }
}]
```

VFV Q4.1:

```
db.[COLLECTION].aggregate([{
 $match: {
  'customer.c_region': {$eq: 'AMERICA'},
  'supplier.s_region': {$eq: 'AMERICA'}
 }
}, {
 $match: {
  $or: [{'part.p_mfgr': {$eq: 'MFGR#1'}},
        {'part.p_mfgr': {$eq: 'MFGR#2'}}]
 }
}, {
 $group: {
  _id: {
```

```
    d_year: '$date.d_year',
    c_nation: '$customer.c_nation'
   },
   profit: {
    $sum: {$subtract: ['$lo_revenue', '$lo_supplycost']}
   }
  }
}]
```

VFV Q4.2:

```
db.[COLLECTION].aggregate([{
 $match: {
  'customer.c_region': {$eq: 'AMERICA'},
  'supplier.s_region': {$eq: 'AMERICA'}
 }
}, {
 $match: {
  $or: [{'date.d_year': {$eq: 1997}},
        {'date.d_year': {$eq: 1998}}]
 }
}, {
 $match: {
  $or: [{'part.p_mfgr': {$eq: 'MFGR#1'}},
        {'part.p_mfgr': {$eq: 'MFGR#2'}}]
 }
}, {
 $group: {
  _id: {
   d_year: '$date.d_year',
   s_nation: '$supplier.s_nation',
   p_category: '$part.p_category'
  },
  profit: {
   $sum: {$subtract: ['$lo_revenue', '$lo_supplycost']}
  }
 }
}]
```

VFV Q4.3:

```
db.[COLLECTION].aggregate([{
 $match: {
  'customer.c_region': {$eq: 'AMERICA'},
  'supplier.s_nation': {$eq: 'UNITED STATES'},
  'part.p_category': {$eq: 'MFGR#14'}
 }
}, {
 $match: {
  $or: [{'date.d_year': {$eq: 1997}},
```

```
        {'date.d_year': {$eq: 1998}}]
 }
}, {
 $group: {
  _id: {
   d_year: '$date.d_year',
   s_city: '$supplier.s_city',
   p_brand1: '$part.p_brand1'
  },
  profit: {
   $sum: {$subtract: ['$lo_revenue', '$lo_supplycost']}
  }
 }
}]
```

Para finalizar os comandos utilizados para o banco de dados MongoDB são exibidos a seguir as consultas para as VMs:

VM Q1.1:

```
db.[COLLECTION].aggregate([{
 $match: {
  '_id.d_year': {'$eq': 1993},
  '_id.lo_discount': {'$in': [1, 3]},
  'id.lo_quantity': {'$lt': 25}
 }
}, {
 $group: {
  '_id': {
    'd_year': '$_id.d_year'
  },
  'revenue': {'$sum': '$revenue'}
  }
}]
```

VM Q1.2:

```
db.[COLLECTION].aggregate([{
 $match: {
  '_id.lo_discount': {'$in': [4, 6]},
  '_id.lo_quantity': {'$in': [26, 34]},
  '_id.d_yearmonthnum': {'$eq': 199401}
 }
}, {
 $group: {
  '_id': {
   'd_yearmonthnum': '$_id.d_yearmonthnum'
  },
  'revenue': {'$sum': '$revenue'}
 }
```

```
}]
```

VM Q1.3:

```
db.[COLLECTION].aggregate([{
 $match: {
  '_id.lo_quantity': {'$in': [26, 35]},
  '_id.lo_discount': {'$in': [5, 7]},
  '_id.d_year': {'$eq': 1994},
  '_id.d_weeknuminyear': {'$eq': 6}
 }
}, {
 $group: {
  '_id': {
   'd_year': '$_id.d_year'
  },
  'revenue': {'$sum': '$revenue'}
 }
}]
```

VM Q2.1:

```
db.[COLLECTION].aggregate([{
 $match: {
  '_id.p_category': {'$eq': 'MFGR#12'},
  '_id.s_region': {'$eq': 'AMERICA'}
 }
}, {
 $sort: {
  '_id.d_year': 1
 }
}]
```

VM Q2.2:

```
db.[COLLECTION].aggregate([{
 $match: {
  '_id.p_brand1': {$in: ['MFGR#2221', 'MFGR#2228']},
  '_id.s_region': {$eq: 'ASIA'}
 }
}, {
 $sort: {
  '_id.d_year': 1
 }
}]
```

VM Q2.3:

```
db.[COLLECTION].aggregate([{
 $match: {
  '_id.p_brand1': {$eq: 'MFGR#2221'},
  '_id.s_region': {$eq: 'EUROPE'}
```

```
 }
}, {
 $sort: {
  '_id.d_year': 1
 }
}]
```

VM Q3.1:

```
db.[COLLECTION].aggregate([{
 $match: {
  '_id.d_year': {$in: [1992, 1997]},
  '_id.c_region': {$eq: 'ASIA'},
  '_id.s_region': {$eq: 'ASIA'}
 }
}, {
 $project: {
  '_id.c_region': 0,
  '_id.s_region': 0
 }
}, {
 $sort: {
  d_year: 1,
  revenue: -1,
  _id: 1
 }
}]
```

VM Q3.2:

```
db.[COLLECTION].aggregate([{
 $match: {
  '_id.d_year': {$in: [1992, 1997]},
  '_id.c_nation': {$eq: 'UNITED STATES'},
  '_id.s_nation': {$eq: 'UNITED STATES'}
 }
}, {
 $project: {
  '_id.s_nation': 0,
  '_id.c_nation': 0
 }
}, {
 $sort: {
  d_year: 1,
  revenue: -1
 }
}]
```

VM Q3.3:

```
db.[COLLECTION].aggregate([{
```

```
 $match: {
  '_id.d_year': { $in: [1992, 1997]}
 }
}, {
 $match: {
  $or: [{'_id.c_city': {$eq: 'UNITED KI1'}},
        {'_id.c_city': {$eq: 'UNITED KI5'}}]
 }
}, {
 $match: {
  $or: [{'_id.s_city': {$eq: 'UNITED KI1'}},
        {'_id.s_city': {$eq: 'UNITED KI5'}}]
 }
}, {
 $sort: {
  d_year: 1,
  revenue: -1
 }
}]
```

VM Q3.4:

```
db.[COLLECTION].aggregate([{
 $match: {
  '_id.d_yearmonth': 'Dec1997'
 }
}, {
 $match: {
  $or: [{'_id.c_city': {$eq: 'UNITED KI1'}},
        {'_id.c_city': {$eq: 'UNITED KI5'}}]
 }
}, {
 $match: {
  $or: [{'_id.s_city': {$eq: 'UNITED KI1'}},
        {'_id.s_city': {$eq: 'UNITED KI5'}}]
 }
}, {
 $sort: {
  d_year: 1,
  revenue: -1
 }
}]
```

VM Q4.1:

```
db.[COLLECTION].aggregate([{
 $match: {
  '_id.c_region': {'$eq': 'AMERICA'},
  '_id.s_region': {'$eq': 'AMERICA'}
 }
```

```
}, {
 $match: {
  '$or': [{'_id.p_mfgr': {'$eq': 'MFGR#1'}},
          {'_id.p_mfgr': {'$eq': 'MFGR#2'}}]
 }
}, {
 $project: {
  '_id.d_year': 1,
  '_id.c_nation': 1,
  'profit': 1
 }
}, {
 $sort: {
  'd_year': 1,
  'profit': -1
 }
}]
```

VM Q4.2:

```
db.[COLLECTION].aggregate([{
 $match: {
  '_id.c_region': {'$eq': 'AMERICA'},
  '_id.s_region': {'$eq': 'AMERICA'}
 }
}, {
 $match: {
  '$or': [{'_id.p_mfgr': {'$eq': 'MFGR#1'}},
          {'_id.p_mfgr': {'$eq': 'MFGR#2'}}]
 }
}, {
 $match: {
  '$or': [{'_id.d_year': {'$eq': 1997}},
          {'_id.d_year': {'$eq': 1998}}]
 }
}, {
 $project: {
  '_id.d_year': 1,
  '_id.s_nation': 1,
  '_id.p_category': 1,
  'profit': 1
 }
}]
```

VM Q4.3:

```
db.[COLLECTION].aggregate([{
 $match: {
  '_id.c_region': {'$eq': 'AMERICA'},
  '_id.s_nation': {'$eq': 'UNITED STATES'},
```

```
  '_id.p_category': {'$eq': 'MFGR#14'}
 }
}, {
 $match: {
  '$or': [{'_id.d_year': {'$eq': 1997}},
          {'_id.d_year': {'$eq': 1998}}]
 }
}, {
 $project: {
  '_id.d_year': 1,
  '_id.s_city': 1,
  '_id.p_brand1': 1,
  'profit': 1
 }
}]
```