MINISTÉRIO DA DEFESA
EXÉRCITO BRASILEIRO
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA
CURSO DE MESTRADO EM SISTEMAS E COMPUTAÇÃO

RAPHAEL AUGUSTO DA SILVA NUNES SORANSSO

UM ESTUDO DO IMPACTO DA MODELAGEM DE DADOS NO DESEMPENHO DE CONSULTAS NOS SGBD NOSQL ORIENTADOS A DOCUMENTOS

Rio de Janeiro
2017

INSTITUTO MILITAR DE ENGENHARIA

RAPHAEL AUGUSTO DA SILVA NUNES SORANSSO

UM ESTUDO DO IMPACTO DA MODELAGEM DE DADOS NO DESEMPENHO DE CONSULTAS NOS SGBD NOSQL ORIENTADOS A DOCUMENTOS

Dissertação de Mestrado apresentada ao Curso de Mestrado em Sistemas e Computação do Instituto Militar de Engenharia, como requisito parcial para a obtenção do título de Mestre em Ciências em Sistemas e Computação.

Orientadora: Prof[a]. Maria Cláudia Reis Cavalcanti - D.Sc.

Rio de Janeiro
2017

INSTITUTO MILITAR DE ENGENHARIA
Praça General Tibúrcio, 80 - Praia Vermelha
Rio de Janeiro - RJ CEP 22290-270

004 Soransso, Raphael Augusto da Silva Nunes
S713e Um Estudo do Impacto da Modelagem de Dados no Desempenho de Consultas nos SGBD NoSQL Orientados a Documentos / Raphael Augusto da Silva Nunes Soransso, orientado por Maria Cláudia Reis Cavalcanti - Rio de Janeiro: Instituto Militar de Engenharia, 2017.

174p.: il.

Dissertação (mestrado) - Instituto Militar de Engenharia, Rio de Janeiro, 2017.

1. Curso de Sistemas e Computação - teses e dissertações. 1. Benchmark. 2. NoSQL. 3. Modelagem de Dados. I. Cavalcanti, Maria Cláudia Reis . II. Título. III. Instituto Militar de Engenharia.

INSTITUTO MILITAR DE ENGENHARIA

RAPHAEL AUGUSTO DA SILVA NUNES SORANSSO

UM ESTUDO DO IMPACTO DA MODELAGEM DE DADOS NO DESEMPENHO DE CONSULTAS NOS SGBD NOSQL ORIENTADOS A DOCUMENTOS

Dissertação de Mestrado apresentada ao Curso de Mestrado em Sistemas e Computação do Instituto Militar de Engenharia, como requisito parcial para a obtenção do título de Mestre em Ciências em Sistemas e Computação.

Orientadora: Prof[a]. Maria Cláudia Reis Cavalcanti - D.Sc.

Aprovada em 02 de Fevereiro de 2017 pela seguinte Banca Examinadora:

Prof[a]. Maria Cláudia Reis Cavalcanti - D.Sc. do IME - Presidente

---

Prof. Ronaldo Ribeiro Goldschmidt - D.Sc. do IME

Prof[a]. Fernanda Araujo Baião Amorim - D.Sc. da UNIRIO

Rio de Janeiro
2017

À Marinha do Brasil pela formação e pela oportunidade de crescimento pessoal e profissional.

AGRADECIMENTOS

Agradeço à minha esposa Renata Biangolino Benício, pelo amor, companherismo, incentivo e motivação desde o momento que nos conhecemos.

Ao meus pais, irmãos e irmãs, que sempre me motivaram no meu caminho, mesmo contrariados pela saudade, que é reciproca.

À família da minha esposa, que agora é minha família, e me incentiva a seguir confiante na minha jornada.

Aos meus amigos, que tornaram os momentos de descanso entre os estudos revigorantes.

A todos estudantes e professores do IME, que me ajudaram e me apoiaram durante o curso de mestrado.

Em especial à minha Professora Orientadora, D.Sc. Maria Cláudia Reis Cavalcanti, por sua dedicação, disponibilidade e atenção.

“Amai ao teu próximo como a ti mesmo. ”

JESUS CRISTO

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

SGBD	Sistemas de Gerenciamento de Banco de Dados
SSB	Star Schema Benchmark
TPC-DS	Transaction Processing Performance Council- Decision Support
XML	Extensible Markup Language
JSON	JavaScript Object Notation
BSON	Binary JSON
OLAP	Online Analytical Processing
OLTP	Online Transaction Processing
CAP	Consistency, Availability and Partition Tolerance
ES	ElasticSearch
JVM	Máquina Virtual do Java
ACID	Atomicidade, Preservação da consistência, Isolamento e Durabilidade
SCR	Seletividade dos Campos Raízes de uma Coleção em uma Consulta
ER	Entidade-Relacionamento
EER	Enhanced Entity–Relationship
BDO	Banco de dados de Objeto
OMT	Object Modeling Technique
GCOLE	Garbage Colector Overhead Limit Exceed

RESUMO

Os SGBD NoSQL surgiram como solução alternativa às limitações dos SGBD Relacionais, com relação ao gerenciamento do crescente volume de dados, e ao seu tratamento distribuído. Esses sistemas além de possuir uma maior facilidade em distribuir os seus dados, permitem uma maior flexibilidade de esquema e são mais tolerantes à inconsistência. Da mesma forma que nos SGBD relacionais, a modelagem lógica dos dados pode influenciar significativamente o desempenho neste tipo de SGBD. Em especial, quando o objetivo é atender aplicações analíticas.

Apesar de existirem alguns trabalhos que buscam orientar a modelagem de dados nos SGBD NoSQL e trabalhos que realizam a avaliação da performance desses sistemas, até onde foi possível investigar, não foram encontrados na literatura trabalhos que demonstrem o quanto as diferentes formas de modelagem dos dados, e mais especificamente da modelagem dos agregados nos SGBD orientados a documentos, podem influenciar no desempenho das consultas neste tipo de sistema. Sem tais estudos, torna-se difícil identificar a melhor alternativa de modelagem de agregados, de forma a melhorar o desempenho das consultas.

Assim, o presente trabalho tem como objetivo preencher esta lacuna e fornecer aos administradores de bancos de dados, orientações de como os dados podem ser modelados de forma a obter melhor desempenho em SGBD orientados a documentos, para consultas analíticas. Para tanto, foram realizados experimentos em cinco SGBD orientados a documentos, utilizando um conjunto de dados, oriundo de um conhecido *benchmark* voltado para aplicações de suporte à decisão. A partir dos resultados dos experimentos foi possível propor heurísticas de modelagem de dados, em algumas das implementações de SGBD estudadas, que foram validadas em outro conjunto de dados oriundo de outro *benchmark* com objetivos similares ao primeiro.

No final do trabalho, foram realizados ainda, um estudo complementar sobre o impacto da modelagem dos agregados na performance das consultas, ao utilizar o recurso de índices, e uma comparação de desempenho entre as implementações estudadas.

ABSTRACT

NoSQL DBMS emerged to address the Relational DBMS limitations with respect to the management of large volumes of data in distributed environments. Similarly to the Relational DBMS, the logical modelling on this kind of database system, can significantly influence its performance, especially when its focus is on answering analytical queries.

Whereas there are some works that provide guidance to the data modelling or that focus on measuring the performance of NoSQL DBMS, there is a lack of works that demonstrate how much the different ways of data modelling, mainly the aggregate modelling, influence the query performance on this kind of document-store database system. Due to this absence, it becomes difficult to identify the best data modelling alternative, i.e., that would improve query performance.

Thus, the present work aims to fulfill this gap and provide data modelling orientation to the database administrators, to increase the performance of document store databases on analytical queries. For this to happen, experiments on five document oriented NoSQL DBMS were performed, using a dataset originated from a well known benchmark for decision support applications. From the experiments results it was possible to propose data modelling heuristics for some of the studied databases. These heuristics were then validated on another dataset, originated from another benchmark with similar objectives of the first one.

At the end of the work, two complementary studies were done: on the aggregate modelling impact on query perfomance, when indexes are used, and a performance comparison among some of the studied DBMS implementations.

1 INTRODUÇÃO

À medida em que a Internet se expandiu e com ela a quantidade de dados trafegados, diversos termos como Web 2.0, Big Data e Internet das Coisas passaram a ser utilizados para definir este novo mundo. Uma característica que essas definições possuem em comum é a necessidade de gerenciar esse enorme volume dados.

Até o final da década de 90, os principais sistemas de gerenciamento de banco de dados (SGBD) das empresas eram os relacionais. Porém à medida em que o volume de dados cresceu, aumentou também a complexidade no seu gerenciamento, devido principalmente à dificuldade de distribuir os dados neste tipo de SGBD (FOWLER; SADALAGE, 2013) (BRITO, 2011). Nesse contexto, surgiram novos tipos de SGBD, conhecidos como SGBD NoSQL.

A forma com que os dados são organizados nos SGBD NoSQL, permite que dados relacionados fiquem armazenados nas mesmas estruturas, o que facilita a sua distribuição em diferentes nós, pois os dados relacionados podem residir nos mesmos nós. Aliado a este fator, uma característica que faz com que os dados sejam distribuídos mais facilmente nos SGBD NoSQL é que estes são mais tolerantes à inconsistência (característica conhecida como consistência eventual), permitindo que os dados em servidores diferentes não necessitem estar consistentes o tempo todo (FOWLER; SADALAGE, 2013). Esta possibilidade de distribuir os dados com mais facilidade é o principal motivo da ascensão dos SGBD NoSQL.

Nos SGBD NoSQL, diferentemente dos SGBD relacionais, é possível ainda adicionar dados aos bancos sem que haja necessidade de se definir, de antemão, quaisquer mudanças na estrutura de organização dos dados (FOWLER; SADALAGE, 2013), o que traz aos SGDB NoSQL uma maior capacidade de lidar com dados não uniformes.

No mundo NoSQL existem variadas formas de se representar os dados, formas estas conhecidas como modelos de dados, onde atualmente entre os modelos mais utilizados se situa o modelo orientado a documentos, existindo, pelo menos 35 implementações deste modelo de dados[1]. A estrutura de organização dos dados neste modelo é conhecida como agregado, que é definido como um conjunto de objetos relacionados que precisam ser tratados como uma unidade (FOWLER; SADALAGE, 2013).

Segundo Navathe (2014a), dentre as técnicas de ajuste de um banco de dados rela-

[1] http://db-engines.com/en/ranking/document+store

cional para melhorar o desempenho de consultas, está a alteração do projeto lógico do banco de dados, que pode consistir em desnormalizar tabelas ou mapear este esquema para um novo conjunto de tabelas físicas e índices. Da mesma forma que nos SGBD relacionais os dados nos SGBD NoSQL também podem ser modelados de formas distintas e esta modelagem pode ter impactos significativos no desempenho desses sistemas (HOBERMAN, 2014). Em virtude dos SGBD NoSQL, mais especificamente os orientados a documentos, utilizarem uma organização dos dados orientada a agregados, a variação da sua modelagem é umas da formas possíveis de se obter melhor desempenho das consultas.

Um tipo de sistema que utiliza os SGBD como forma de gerenciar os seus dados, são as aplicações analíticas. Neste tipo de aplicação a quantidade de dados a ser gerenciada pode fazer com que os SGBD NoSQL também sejam uma opção para a melhoria do seu desempenho (CHEVALIER et al., 2015).

Nos SGBD NoSQL a modelagem dos dados é realizada de forma a resolver problemas específicos e como nas aplicações analíticas o desempenho de consultas é de extrema importância, a modelagem dos dados para este tipo de aplicação deve ser voltada para este objetivo. Normalmente, neste tipo de aplicação, mais de uma entidade, conhecidas como tabelas no modelo relacional, são consultadas ao mesmo tempo (O'NEIL et al., 2009). Devido aos SGBD NoSQL serem aversos a operações de junção (FOWLER; SADALAGE, 2013), organizar os dados relacionados de forma que estes estejam em um único agregado, permite uma maior eficiência nas consultas analíticas (HOBERMAN, 2014).

Esta forma de organizar os dados é conhecida como dados aninhados e da mesma forma que nos SGBD XML nativos, a modelagem nos SGBD NoSQL orientados a documentos, utilizando agregados com dados aninhados pode ser realizada de diversas formas (NAVATHE, 2014b).

Nos SGBD NoSQL orientados a documentos, utilizar uma certa modelagem de agregado pode proporcionar um bom desempenho para um número limitado de consultas de uma aplicação analítica, mas prejudicar o desempenho de outras. Em virtude dos SGBD NoSQL serem mais relaxados quanto à consistência, este fato permite lidar com uma maior redundância dos dados. Assim, a replicação de dados em mais de uma modelagem de agregados, pode ser usada em prol de beneficiar um número maior de consultas, e trazer ganhos significativos para as aplicações analíticas.

Por fim, dadas as variadas implementações deste tipo de SGBD, é possível que o impacto da modelagem dos agregados seja diferente em cada implementação, de forma que as escolhas de modelagens para um SGBD podem ser diferentes das escolhas para um outro SGBD.

Por se tratarem de uma tecnologia recente, até onde foi possível investigar, não foram encontrados estudos que buscassem compreender o comportamento dos SGBD orientados a documentos, à medida em que a modelagem de agregados é alterada. Logo, devido à ausência do entendimento do comportamento desses bancos em diferentes modelagens de agregados, não existe atualmente algum tipo de orientação de como a variação da modelagem dos agregados pode ser utilizada, de forma que as diversas implementações tenham um melhor desempenho nas consultas.

Entre os trabalhos encontrados que envolvem avaliações de desempenho de SGBD NoSQL (CARNIEL et al., 2012b), (ABRAMOVA et al., 2014), (ABUBAKAR et al., 2014), (PHAN et al., 2014), (GANDINI et al., 2014), entre outros, todos, até o presente momento, utilizam somente uma modelagem dos agregados, onde a comparação de desempenho entre diferentes implementações utilizando diferentes modelagens de agregados ainda é ausente na comunidade científica.

Portanto, é no contexto do entendimento do comportamento das diversas implementações, à medida em que a modelagem dos agregados pode ser modificada e na orientação de como a modelagem de agregados pode ser utilizada de forma a otimizar consultas analíticas, que se situa este trabalho.

Complementarmente ao estudo principal deste trabalho, será realizado também um estudo sobre o impacto da modelagem de dados e dos benefícios da utilização do recurso de índices no desempenho de consultas.

A partir da análise do comportamento dos diferentes sistemas de bancos de dados será possível ainda realizar uma comparação de desempenho entre diferentes implementações de SGBD orientados a documentos, sem utilizar índices e utilizando índices, de forma a auxiliar na escolha entre uma implementação ou outra.

1.1 MOTIVAÇÃO

Os primeiros e mais conhecidos SGBD NoSQL criados foram o Big Table da Google, o Dynamo da Amazon e o Cassandra do Facebook (FOWLER; SADALAGE, 2013). Em virtude desses sites estarem entre os mais acessados[2] e por estes utilizarem esses SGBD como principal ferramenta de gerenciamento dos seus dados, este fato demonstra a importância dos SGBD NoSQL no cenário atual.

Atualmente existem 111 SGBD NoSQL[3], que seguem as 4 abordagens de modelos

[2]http://exame.abril.com.br/tecnologia/os-10-sites-mais-visitados-do-mundo/
[3]http://nosql-database.org

de dados mais conhecidas: 14 são do modelo de dados família de colunas, 26 do modelo orientado a documentos, 50 do modelo chave-valor e 21 do modelo orientado a grafos.

Essa diversidade motivou a realização de estudos que orientassem as diversas organizações públicas e privadas na organização dos dados para estes sistemas, bem como na escolha de qual implementação melhor se encaixa nas necessidades do usuário.

Entre os diversos modelos de SGBD NoSQL existentes, um dos mais utilizados e conhecidos na atualidade é o modelo orientado a documento. Os SGBD NoSQL orientados a documento possuem uma estrutura de organização dos dados diferente dos relacionais, porém da mesma forma que a modelagem de dados nos relacionais afeta o desempenho das consultas nestes, a modelagem dos agregados nos SGBD orientados a documentos também pode influenciar o desempenho de suas consultas (HOBERMAN, 2014). Além disso, dada a variedade de implementações deste tipo de SGBD, as modelagens que proveem melhor desempenho para uma podem não ser as mesmas para outra implementação.

Um tipo de aplicação que se beneficia da melhora do desempenho de consultas a partir de diferentes modelagens é a aplicação analítica. Este tipo de aplicação muitas vezes exige a manutenção de redundância de dados para atender melhor suas consultas *ad-hoc*. Neste contexto, é bastante útil a indicação das modelagens que melhor atendem às consultas críticas.

Entretanto, há ainda poucos estudos, como o de Scabora et al. (2016), que analisam o comportamento dos SGBD NoSQL sobre diferentes modelagens de dados e apesar de existirem trabalhos, como os de Abramova et al. (2014), Gandini et al. (2014), Abubakar et al. (2014) e Phan et al. (2014), que realizam a comparação de desempenho entre SGBD NoSQL, em operações de escrita, leitura e atualização, por vezes variando o número de nós e medindo o número de transações por tempo, os dados utilizados nestes trabalhos possuem uma organização simples, não variando a sua modelagem.

1.2 OBJETIVOS E HIPÓTESE

Tendo em vista a ausência de estudos a respeito da influência da modelagem de dados no desempenho das consultas nos SGBD orientados a documentos, este trabalho tem como objetivo realizar um estudo do impacto no desempenho de consultas em diferentes SGBD NoSQL orientados a documentos quando utilizadas diferentes modelagens de dados.

Este trabalho possui como objetivos secundários prover heurísticas de modelagem de dados para os diferentes SGBD estudados, servindo como uma orientação de como os dados podem ser organizados de forma a obter um melhor desempenho das consultas em

cada SGBD, utilizando a forma padrão de consulta nesses sistemas, sem a criação manual de índices, realizar um estudo do impacto da modelagem de agregados no desempenho das consultas utilizando índices e ainda servir como referência para comparação de desempenho entre os bancos de dados estudados, em consultas realizadas em dados modelados de diferentes formas, em estruturas mais complexas.

A hipótese deste trabalho é que a modelagem de agregados pode influenciar de formas distintas o desempenho das consultas realizadas nos diferentes implementações de SGBD orientados a documentos. Desta forma, os administradores de bancos de dados devem ser orientados na escolha de uma ou mais modelagens de agregados, nas diferentes implementações, de forma a obter um maior desempenho nas consultas analíticas.

1.3 CONTRIBUIÇÕES

Apesar do crescente número de trabalhos a respeito dos SGBD NoSQL, até onde foi possível investigar, não foram encontrados estudos demostrando o desempenho de consultas dos SGBD NoSQL, sobre diferentes modelagens de dados nos SGBD orientados a documentos, fazendo com que este trabalho se diferencie dos demais, trazendo uma nova perspectiva sobre a influência da modelagem de dados nos SGBD orientados a documentos.

Sabendo da influência das modelagens de dados sobre os diferentes bancos de dados, a escolha de qual modelagem de dados utilizar para melhorar o desempenho das consultas não é trivial, sendo necessário o estabelecimento um método a ser seguido que oriente os administradores de bancos de dados de qual modelagem utilizar. Portanto este trabalho também possui como contribuição a construção de heurísticas voltadas à orientação dos administradores de bancos de dados para a escolha de qual a melhor modelagem para os seus dados.

A terceira contribuição deste trabalho está relacionada com um estudo da possibilidade de utilizar índices em estruturas de dados mais complexas e no impacto da utilização deste recurso nas diferentes implementações, tanto no que diz respeito ao aumento do desempenho das consultas, quanto no que diz respeito ao impacto da modelagem de dados nos SGBD orientados a documentos utilizando estes recursos.

Como contribuição final, podemos citar ainda, a comparação de desempenho de diferentes implementações deste tipo de SGBD, tanto utilizando índices quanto não utilizando índices, que poderá ser utilizada pelos administradores de banco de dados como parâmetro de escolha entre uma implementação e outra.

1.4 ORGANIZAÇÃO DA DISSERTAÇÃO

Este trabalho está dividido em oito capítulos. Neste capítulo foi apresentada a introdução deste trabalho e no capítulo dois serão apresentados os conceitos básicos que nortearão o presente trabalho. No terceiro capítulo serão apresentados os trabalhos relacionados ao tema desta dissertação. No capítulo quatro serão apresentados e discutidos os resultados do experimento inicial e serão propostas heurísticas de modelagem de agregados para os SGBD estudados e no quinto capítulo as heurísticas propostas serão validadas em um conjunto de dados diferente do utilizado nos experimentos iniciais. No capítulo seis será discutido o impacto da modelagem nos SGBD ao utilizar índices e no capítulo sete o desempenho dos SGBD estudados serão comparados. No último capítulo será apresentada a conclusão do presente trabalho.

2 CONCEITOS BÁSICOS E DESCRIÇÃO DAS TECNOLOGIAS USADAS

Este capítulo apresenta alguns conceitos básicos para entendimento do trabalho. Inicialmente são explicados alguns conceitos relacionados aos SGBD NoSQL. A seguir, são descritas as tecnologias utilizadas neste trabalho, abrangendo algumas implementações de SGBD orientados a documentos, fazendo uma breve comparação entre as suas características e uma breve explanação sobre os *Benchmarks* utilizados nos experimentos realizados.

2.1 SGBD NOSQL

Inicialmente o termo NoSQL se referia aos SGBD que não utilizavam a linguagem SQL, porém seu significado sofreu uma evolução, passando a ser utilizado como "Not Only SQL" devido a alguns SGBD NoSQL utilizarem uma linguagem similar ao SQL. Ainda mais adiante este termo passou a ser utilizado de forma mais genérica para referenciar os SGBD não-relacionais.

Segundo Brito (2011), o principal motivo que levou ao surgimento dos SGBD NoSQL, foi o crescimento do volume de dados a serem armazenados e com isso a necessidade de escalonar os dados mais facilmente do que os SGBDs relacionais. Portanto, os SGBDs NoSQL foram uma solução para facilitar o escalonamento do grande volume de dados gerados pela atual conjectura do mundo Big Data. Esta característica se deve ao fato desse tipo de SGBD permitir uma estrutura de dados mais flexível que os bancos de dados relacionais.

Diferentemente dos bancos relacionais que armazenam os dados em tuplas dentro tabelas que contém dados relacionados, parte dos SGBD NoSQL armazenam os dados na forma de agregados, permitindo que os dados relacionados sejam acessados em conjunto, sem a necessidade de se realizar junções, logo, dados relacionados podem ser arquivados nos mesmos nós com maior facilidade. Aliado a este fator, uma característica que faz com que os dados sejam distribuídos mais facilmente nos SGBD NoSQL é que estes são mais tolerantes à consistência, dando maior liberdade para os dados, de forma que ao serem armazenados em servidores diferentes, não necessitem estar consistentes o tempo todo.

Somado a esta facilidade de escalonar os dados, os SGBD NoSQL permitem também uma maior flexibilidade no esquema de dados, fazendo com que seja possível adicionar

campos aos registros de dados sem ter que definir alguma alteração na estrutura dos dados (FOWLER; SADALAGE, 2013).

Outra característica, porém de menor importância que pode ser considerada relevante para o início da utilização dos SGBD NoSQL é a chamada incompatibilidade de impedância existente entre os dados dos SGBDs relacionais e os dados arquivados em memória. Pois a forma com que os dados são estruturados na memória permitem uma organização mais complexa do que simples linhas e colunas. (FOWLER; SADALAGE, 2013).

A maioria dos SGBDs NoSQL são de código aberto, porém quase todos possuem uma versão paga que possui algum recurso a mais. O MongoDB, por exemplo, possui uma versão avançada conhecida como MongoDB *Enterprise*, onde o usuário pode contar com alguns recursos não disponíveis na versão gratuita, como por exemplo, o suporte técnico direto dos fornecedores do SGBD, a disponibilização de uma ferramenta de gerenciamento de performance, com opções avançadas de segurança, entre outras opções.

Existem atualmente quatro modelos de bancos de dados NoSQL: Chave-Valor, Orientado a Documento, Orientado a Colunas e Orientado a Grafos. Este trabalho se limitará a estudar somente os SGBD orientados a documentos, pois a forma de organizar os agregados neste modelo é diferente dos outros modelos e o estudo do impacto da modelagem em outros modelos tornaria o presente trabalho muito extenso.

As propriedades ACID, segundo Navathe (2014a) são as propriedades desejáveis das transações dos bancos de dados relacionais a saber: Atomicidade, Consistência, Isolamento e Durabilidade.

Os SGBDs NoSQL, devido às suas características e à finalidade para qual estes foram criados, não seguem as propriedades ACID da mesma forma que os SGBD relacionais, porém seguem o teorema CAP.

Este teorema foi criado por Eric Brewer e segundo Brito (2011), consiste na teoria de que em um sistema distribuído entre as propriedades Consistência, Disponibilidade e Tolerância à Partição[4], somente é possível garantir duas delas simultaneamente. Porém, de acordo com Fowler e Sadalage (2013), um sistema particionado pode balancear consistência com disponibilidade, o que significa que é possível obter um sistema particionado com um pouco de disponibilidade e também com um pouco de consistência, e isto irá variar conforme estão configurados os servidores e como os dados são replicados ou distribuídos.

A seguir apresentamos alguns conceitos relacionados a SGBD NoSQL, em especial os orientados a documentos, que são o foco deste trabalho.

[4]http://www.julianbrowne.com/article/viewer/brewers-cap- theorem

2.1.1 AGREGADOS

Diferentemente dos SGBD relacionais os dados nos SGBD NoSQL não são arquivados em tabelas que contém tuplas e colunas, mas em uma forma chamada "orientada a agregados" (FOWLER; SADALAGE, 2013). Esta forma de organização permite que os dados relacionados possam ser armazenados dentro de um mesmo agregado.

Dentre os modelos de dados NoSQL existentes, os modelos colunar, orientados a documento e o chave-valor são fortemente orientados a agregados, sendo que o modelo orientado a grafo não segue uma orientação agregada. Em cada modelo de SGBD NoSQL, os agregados são representados de forma diferente, onde no modelo orientado a documento, que é o foco deste estudo, cada documento é considerado um agregado.

Segundo Evans (2004), os agregados são definidos como unidades de manipulação de dados e de gerenciamento de consistência e que apesar dos bancos NoSQL não realizarem operações atômicas em muitos agregados, as operações dentro de um único agregado são atômicas (FOWLER; SADALAGE, 2013).

Conforme (HOBERMAN, 2014) é possível organizar os agregados em um SGBD orientado a documento de duas formas distintas, utilizando referências ou aninhando os dados. O uso de referências em um SGBD orientado a documento ocorre de forma similar a um SGBD relacional, exceto pelo fato de que nos orientados a documentos não é necessário definir quais campos serão utilizados como chaves estrangeiras, porém diferentemente dos relacionais, os SGBD NoSQL, de forma geral não lidam bem com junção (FOWLER; SADALAGE, 2013). Assim, a utilização de referências em SGBD NoSQL só se justifica quando os dados de cada coleção de agregados são acessados e manipulados separadamente (HOBERMAN, 2014).

Portanto, em virtude deste estudo ter como foco a avaliação do impacto da modelagem de dados em consultas voltadas para aplicações de apoio à decisão, existe a necessidade de que os agregados utilizados neste trabalho possuam os dados relacionados aninhados e não utilize referências a outros agregados, de forma a evitar que seja necessária a realização da operação de junção entre agregados diferentes, seguindo o primeiro passo da heurística proposta por (HOBERMAN, 2014), que sugere que os dados consultados juntos devem ser aninhados.

Para melhor entendermos a organização de um agregado, que equivale a um documento em um SGBD orientado a documentos (BUGIOTTI et al., 2014), suponhamos um esquema de dados relacional, que pode ser visualizado por meio da figura 2.1 e das relações na figura 2.2. Neste esquema, a tabela de fato *Lineorder* possui um relacionamento N:1

com as tabelas de dimensão *Customer*, *Supplier*, *Orderdate* e *Part* conforme (HEUSER, 2009), cujas chaves estrangeiras das tabelas de dimensão são respectivamente os atributos *lo_custkey, lo_suppkey, lo_datekey, lo_partkey.*

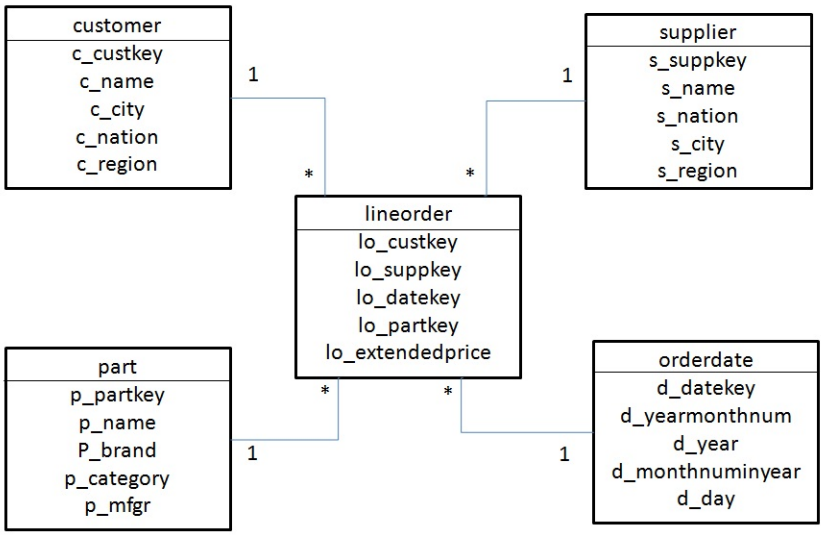

FIG. 2.1: Esquema de dados relacional.

Customer				
c_custkey	c_name	c_city	c_nation	c_region
1	Customer#006503	IRAQ 6	IRAQ	MIDDLE EAST
2	Customer#006504	IRAQ 7	IRAQ	MIDDLE EAST

Lineorder				
lo_custkey	lo_suppkey	lo_datekey	lo_partkey	lo_extendedprice
1	1	1	1	1000
1	2	2	2	2000

Part				
p_partkey	p_name	p_brand	p_category	p_mfgr
1	tomato midnight	MFGR#341	MFGR#34	MFGR#3
2	dark cornflower	MFGR#421	MFGR#42	MFGR#4

Orderdate				
d_datekey	d_yearmonthnum	d_year	d_monthnuminyear	d_day
1	199504	1995	4	1
2	199504	1995	4	2

Supplier				
s_suppkey	s_name	s_nation	s_city	s_region
1	Supplier#001	UNITED STATES	UNITED ST8	AMERICA
2	Supplier#002	ALGERIA	ALGERIA 1	AFRICA

FIG. 2.2: Relações em um banco de dados relacional.

Na figura 2.3, podemos visualizar como os dados deste esquema são representados na forma agregada, a partir de um documento no formato JSON[5]. Nesta figura, podemos observar que os dados relacionados são armazenados dentro de um único documento e o relacionamento entre cada linha da tabela *Lineorder* com as tuplas das outras tabelas

[5]json.org

é representado pelo aninhamento dos dados das tuplas das outras tabelas aos dados da tabela *Lineorder*. Portanto, para cada linha da tabela *Lineorder* é criado um documento, onde os dados aninhados representam os dados relacionados com a linha utilizada como base para o documento.

A chaves primárias e estrangeiras nos documentos apresentados não são necessárias nesta forma de representação, porém recomenda-se a manutenção dessas, de forma que cada dado aninhado possua uma identificação, pois alguns SGBD podem valer-se dessas chaves para otimizar suas consultas.

Documento 1

```
{
"lo_extendedprice": 1000,
"supplier": [{
        "s_name": "Supplier#001",
        "s_nation": "UNITED STATES",
        "s_city": "UNITED ST8",
        "s_region": "AMERICA"}],
"part": [{
        "p_brand1": "MFGR#341",
        "p_category": "MFGR#34",
        "p_name": "tomato midnight",
        "p_mfgr": "MFGR#3"}],
"orderdate": [{
        "d_monthnuminyear": 4,
        "d_yearmonthnum": 199504,
        "d_year": 1995,
        "d_day": 1}],
"customer": [{
        "c_name": "Customer#006503"
        "c_city": "IRAQ    6",
        "c_region": "MIDDLE EAST",
        "c_nation": "IRAQ"}]
}
```

Documento 2

```
{
"lo_extendedprice": 2000,
"supplier": [{
        "s_name": "Supplier#002",
        "s_nation": "ALGERIA",
        "s_city": "ALGERIA  1",
        "s_region": "AFRICA"}],
"part": [{
        "p_brand1": "MFGR#421",
        "p_category": "MFGR#42",
        "p_name": "dark cornflower",
        "p_mfgr": "MFGR#4"}],
"orderdate": [{
        "d_monthnuminyear": 4,
        "d_yearmonthnum": 199504,
        "d_year": 1995,
        "d_day": 2}],
"customer": [{
        "c_name": "Customer#006503"
        "c_city": "IRAQ    6",
        "c_region": "MIDDLE EAST",
        "c_nation": "IRAQ"}]
}
```

FIG. 2.3: Dados representados no formato de agregado.

2.1.2 VARIAÇÃO DA MODELAGEM DOS AGREGADOS

O agregado utilizado como exemplo na seção anterior, figura 2.3, utilizou a tabela de fatos como base para a construção dos agregados. Os dados desta tabela foram inseridos na raiz dos documentos e os dados das tabelas de dimensão foram aninhados aos dados da tabela de fato. Entretanto, é possível modificar a tabela utilizada como base para a construção dos agregados de forma a obter agregados modelados de diferentes formas.

Na figura 2.4 é possível observar uma modelagem de agregado diferente da modelagem da figura 2.3. Nesta modelagem, a tabela que foi utilizada como base para o agregado foi a tabela *Customer*.

Como a tabela *Customer* tem um relacionamento 1:N com a tabela de fatos *Lineorder*, os dados das tuplas da tabela *Lineorder* que possuiam relacionamentos com a tabela base foram aninhados aos dados daquela tabela.

Repare que as duas linhas da tabela *Lineorder*, da figura 2.2, possuem relacionamento com somente uma tupla da tabela *Customer*, logo, no documento que possui na sua raiz os dados do cliente "Customer#006503", haverá duas linhas da tabela *Lineorder* aninhadas aos dados deste cliente. Além disso, a tabela *Lineorder* também possui relacionamentos com as outras tabelas de dimensão, e da mesma forma demostrada na seção anterior, os seus relacionamentos com as outras tabelas de dimensão serão representados pelo aninhamento dos dados daquelas aos dados da tabela *Lineorder*. Portanto, nesta modelagem os dados de cada tupla da tabela *Customer* estarão em um único documento e os dados relacionados a cada tupla estarão aninhados aos seus dados.


```
[{"c_name": "Customer#006503",
  "c_city": "IRAQ     6",
  "c_region": "MIDDLE EAST",
  "c_nation": "IRAQ",
  "lineorder":[{
        "lo_supplycost": 70806,
        "lo_quantity": 8,
        "lo_revenue": 859112,
        "lo_extendedprice": 944080,
        "lo_discount": 9,
        "supplier": [{
              "s_nation": "UNITED STATES",
              "s_city": "UNITED ST8",
              "s_region": "AMERICA"
        }],
        "part": [{
              "p_brand1": "MFGR#3426",
              "p_category": "MFGR#34",
              "p_name": "tomato midnight"
        }],
        "orderdate": [{
              "d_monthnuminyear": 4,
              "d_yearmonthnum": 199504,
              "d_year": 1995
        }]
},
{
  "lo_supplycost": 90033,
  "lo_quantity": 39,
  "lo_revenue": 5704296,
  "lo_extendedprice": 5820711,
  "lo_discount": 2,
  "supplier": [{
        "s_nation": "ALGERIA",
        "s_city": "ALGERIA  1",
        "s_region": "AFRICA"
  }],
  "part": [{
        "p_brand1": "MFGR#4229",
        "p_category": "MFGR#42",
        "p_name": "dark cornflower"
  }],
  "orderdate": [{
        "d_monthnuminyear": 4,
        "d_yearmonthnum": 199504,
        "d_year": 1995
  }]
 }
]
```


FIG. 2.4: Dados representados no formato de agregado utilizando a tabela *Customer* como base para o agregado.

Da mesma forma que a tabela *Customer* foi utilizada como base do agregado representado pela figura 2.4, é possível construir outras três modelagens de agregados, utilizando como base as outras tabelas de dimensão, *Part*, *Supplier* e *Date*. Logo, é possível variar a modelagem do agregado construído a partir do esquema representado pela figura 2.1 de cinco formas diferentes, cada modelagem utilizando uma tabela como base para o

agregado.

No trabalho de (CHEVALIER et al., 2015) é apresentada a formalização de uma modelagem dos agregados de um SGBD NoSQL orientado a documentos a partir de um esquema multidimensional, onde foi utilizado um caso de estudo visando facilitar o entendimento da formalização.

Com base neste trabalho, apresentaremos uma formalização genérica das diferentes modelagens que os agregados podem assumir, ao se migrar os dados a partir de um esquema conceitual multidimensional, da mesma forma que os agregados foram modelados nos exemplos apresentados anteriormente nesta seção:

Seja um esquema multidimensional E definido por (F, D) onde:

- F é a tabela de fato,
- D=$\{D_1, \ldots, D_p\}$ é um conjunto finito de tabelas de dimensão pertencente ao esquema E.

  Uma tabela de dimensão D_i é definida por (N^{D_i}, A^{D_i}) onde:

- N^{D_i} é o nome da tabela de dimensão D_i,
- $A^{D_i} = \{A_1^{D_i}, \ldots, A_n^{D_i}\}$ é um conjunto de atributos da tabela D_i,
- $V_j^{D_i} = \{v_{1j}^{D_i}, ..., v_{nj}^{D_i}\}$ são os valores dos atributos $A_1^{D_i}, \ldots, A_n^{D_i}$ em uma linha j da tabela D_i.

  Uma tabela de fato é definida por (N^F, A^F), onde:

- N^F é o nome da tabela de fatos F,
- $A^F = \{A_1^F, \ldots, A_n^F\}$ é um conjunto de medidas ou atributos da tabela F,
- $V_j^F = \{v_{1j}^F, ..., v_{nj}^F\}$ são os valores dos atributos $A_1^F, \ldots, A_n^F$ em uma linha j da tabela F.

  Em (CHEVALIER et al., 2015), uma coleção de um SGBD orientado a documentos C é definida como um conjunto de documentos C=$\{T_1, ..., T_k\}$ e cada documento T_i é definido por $T_i = \{(Att_i^1, V_i^1), ..., (Att_i^n, V_i^n)\}$, onde Att_i^j é um atributo e V_i^j é um valor que pode assumir duas formas:

  - O valor é atômico, ou

  - O valor é composto, que corresponde a um documento aninhado que é definido por um novo conjunto de pares (atributo, valor).

Naquele trabalho foi apresentada a formalização da modelagem de agregado baseada somente na tabela de fatos. Portanto, neste trabalho iremos apresentar uma formalização da modelagem dos agregados baseados na tabela de fatos e nas tabelas de dimensão, para um conjunto de dados genérico.

A partir de um esquema multidimensional E é possível obter diferentes modelagens de agregados, cada uma baseada em uma diferente tabela existente no esquema da seguinte forma:

Seja $C^F = \{T_1^F, ..., T_x^F, ..., T_k^F\}$ a coleção de documentos cuja modelagem de agregados é baseada na tabela de fatos, nesta modelagem o documento T_x^F pode ser definido como:

$T_x^F = \{(Att_x^{r_1}, V_x^{r_1}), ..., (Att_x^{r_m}, V_x^{r_m}), (Att_x^{D_{(m+1)}}, V_x^{D_{(m+1)}}), ..., (Att_x^{D_p}, V_x^{D_p})\}$, onde:

- o subconjunto $(Att_x^{r_1}, V_x^{r_1}), ..., (Att_x^{r_m}, V_x^{r_m})$ é o conjunto formado pelos atributos $A_1^F, \ldots, A_m^F$ e pelos valores destes atributos em uma linha x da tabela de fatos, ou seja, $v_{1x}^F, ..., v_{mx}^F$, e

- o subconjunto $(Att_x^{D_{(m+1)}}, V_x^{D_{(m+1)}}), ..., (Att_x^{D_p}, V_x^{D_p})$ é o conjunto formado pelos pares de atributos e seus valores compostos, onde $Att_x^{D_k} = N^{D_k}$ e $V_x^{D_{(m+1)}}$ a $V_x^{D_n}$ são documentos aninhados da seguinte forma:

  - $V_x^{D_k} = \{(A_1^{D_k}, v_{1j}^{D_k}), \ldots, (A_n^{D_k}, v_{nj}^{D_k})\}$, onde $(A_1^{D_k}, v_{1j}^{D_k})$ corresponde a um atributo e seu valor em uma linha j de uma tabela de dimensão D_k que possui relacionamento com a linha x de F.

Seja $C^{D_i} = \{T_1^{D_i}, ..., T_y^{D_i}, ..., T_k^{D_i}\}$ uma coleção de documentos cuja modelagem de agregados é baseada em uma tabela de dimensão D_i, nesta modelagem o documento $T_y^{D_i}$ pode ser definido como:

$T_y^{D_i} = \{(Att_y^{r_1}, V_y^{r_1}), ..., (Att_y^{r_m}, V_y^{r_m}), (Att_y^{F_{(m+1)}}, V_y^{F_{(m+1)}})\}$, onde:

- o subconjunto $(Att_y^{r_1}, V_y^{r_1}), ..., (Att_y^{r_m}, V_y^{r_m})$ é o conjunto formado pelos pares de atributos $A_1^{D_i}, \ldots, A_m^{D_i}$ e pelos valores destes atributos em uma linha y da tabela de dimensão D_i, ou seja, $v_{1y}^{D_i}, ..., v_{my}^{D_i}$, e

- o par $(Att_y^{F_{(m+1)}}, V_y^{F_{(m+1)}})$ é formado por um único atributo e seus valores compostos, onde $Att_y^{F_{(m+1)}} = N^F$ e $V_y^{F_{(m+1)}}$ são documentos aninhados da seguinte forma:

– $V_y^{F_{(m+1)}} = T_y^F = \{(A_1^{F_1}, v_{1j}^F), ..., (A_n^{F_k}, v_{nj}^F), (Att^{D_{i'}}, V^{D_{i'}})\}$, onde, os pares $(A_y^{F_1}, v_{1j}^F), ..., (A_y^{F_k}, v_{kj}^F)$ correspondem aos atributos da tabela de fatos e seus respectivos valores em cada linha j da tabela de fato que possui relacionamento com uma linha y da tabela D_i e $D_{i'} = D - \{D_i\}$.

Após a compreensão da definição do conceito de agregado e de como a sua modelagem pode variar, fica mais evidente o objetivo principal deste trabalho, que é avaliar o impacto da variação da modelagem dos agregados, demonstrada nesta seção, no desempenho das consultas em diferentes implementações de banco de dados orientados a documentos.

2.1.3 DESAGREGAÇÃO

Em alguns SGBD orientados a documentos é necessária a realização de uma operação de desagregação dos agregados para que algumas consultas sejam realizadas. Para melhor entendermos esta operação, demonstraremos como esta operação seria realizada de forma simplificada no SGBD OrientDB, em virtude da sua linguagem de consulta ser similar ao SQL.

Suponha que exista uma coleção em um SGBD orientado a documento, cujo agregado representado pela figura 2.4 seja o único documento desta coleção. Se realizarmos uma consulta onde desejamos saber a quantidade de ”tomato midnight” comprado pelo cliente ”Customer#006503”, sem a operação de desagregação no OrientDB ela seria expressa da seguinte forma:

```
SELECT lineorder.lo_quantity
FROM CUSTOMER
WHERE c_name="Customer#006503" AND
lineorder.part.p_name="tomato midnight";
```

O resultado desta operação ao invés de retornar somente o valor 8, relativo ao valor de *lo_quantity*, onde somente a condição de seleção é satisfeita, retornará os valores 8 e 39. Isso ocorre por que o documento completo é retornado como resultado. Como este documento, cuja raiz tem c_name=”Customer#006503”, aninha vários *lineorders*, todos estes acabam sendo considerados, conforme pede a cláusula de projeção da consulta. Portanto, é necessário separar os pedidos aninhados, como se estivessem em documentos separados, para que o resultado retornado considere somente o pedido cujo nome do produto seja ”tomato midnight”.

Esta separação dos campos aninhados dentro de uma lista é chamada de **operação**

de desagregação. No OrientDB esta operação é chamada de operação *unwind*. Ao reescrevermos a mesma consulta na linguagem do OrientDB, utilizando a desagregação esta será expressa da seguinte forma:

```
SELECT line.lo_quantity
FROM (SELECT c_name, c_city, c_region, c_nation,
lineorder as line
FROM CUSTOMER
UNWIND line)
WHERE c_name="Customer#006503" AND
lineo.part.p_name="tomato midnight";
```

Considerando somente a operação de desagregação realizada, o resultado é o apresentado pela figura 2.5. Repare que os pedidos dentro da lista *lineorder* foram separados em dois documentos diferentes, permitindo que a consulta possa selecionar apenas os dados do pedido desejado. Portanto, o resultado retornado da consulta passa a ser somente 8.

Apesar do exemplo acima ter demonstrado a realização da operação de desagregação em somente um documento, a realização da operação de desagregação em uma coleção que possui muitos documentos pode ser feita em muitos documentos, e por isso é considerada uma operação custosa.

Porém, antes de realizar a operação de desagregação é possível realizar uma pré-seleção dos documentos que satisfazem à condição de seleção, de forma a diminuir a quantidade de documentos nos quais esta operação será realizada, ou seja, a consulta acima pode ser reescrita da seguinte forma:

```
SELECT line.lo_quantity
FROM (SELECT c_name, c_city, c_region, c_nation, lineorder as line
FROM CUSTOMER where ”Customer#006503” AND lineorder.part.p_name=”tomato
midnight”
UNWIND line)
WHERE lineo.part.p_name=”tomato midnight”;
```

Portanto, a sub-consulta selecionará os documentos que satisfazem à condição de seleção e a operação de desagregação será realizada somente nesses documentos, podendo levar a uma otimização considerável no desempenho da consulta.

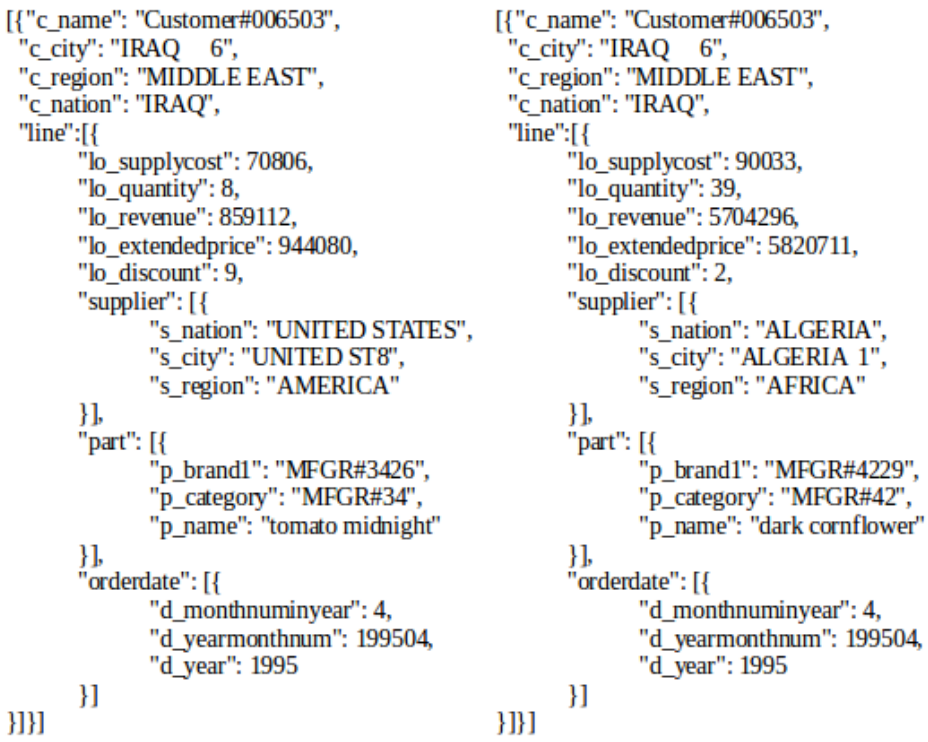

FIG. 2.5: Desagregação do agregado *Customer*.

2.1.4 MAP-REDUCE

Segundo Fowler e Sadalage (2013), Map-Reduce é um padrão que tem como objetivo estabelecer uma forma de organizar o processamento de dados de muitas máquinas de um cluster. Podendo este padrão ser dividido basicamente em duas funções: mapeamento (Map) e redução dos dados (Reduce).

A figura 2.6[6] apresenta um exemplo de como o tratamento de documentos pode se beneficiar deste recurso de processamento paralelo. Nesta figura podemos observar que documentos foram distribuídos para processamento em diferentes nós. Para cada documento que entra, uma **função de mapeamento (map)** realiza uma operação que extrai as palavras do documento e conta o número de vezes que cada palavra apareceu no mesmo, representando um valor para cada palavra. O resultado dessas operações é uma lista de palavras e seus totais respectivos. Esta lista pode ser a entrada de uma **função de redução (reduce)** em outro nó, que realiza uma operação que soma as quantidades de palavras das listas recebidas.

O exemplo ilustra como paralelizar operações de mapeamento, cujo resultado pode ser consumido por operações de redução. As operações podem ser quaisquer, desde que se

[6]http://3.bp.blogspot.com/-uje8RCtirl4/U8bmrMQ-diI/AAAAAAAAAyo/HiF14HvV-ls/s1600/map-reduce-example.png

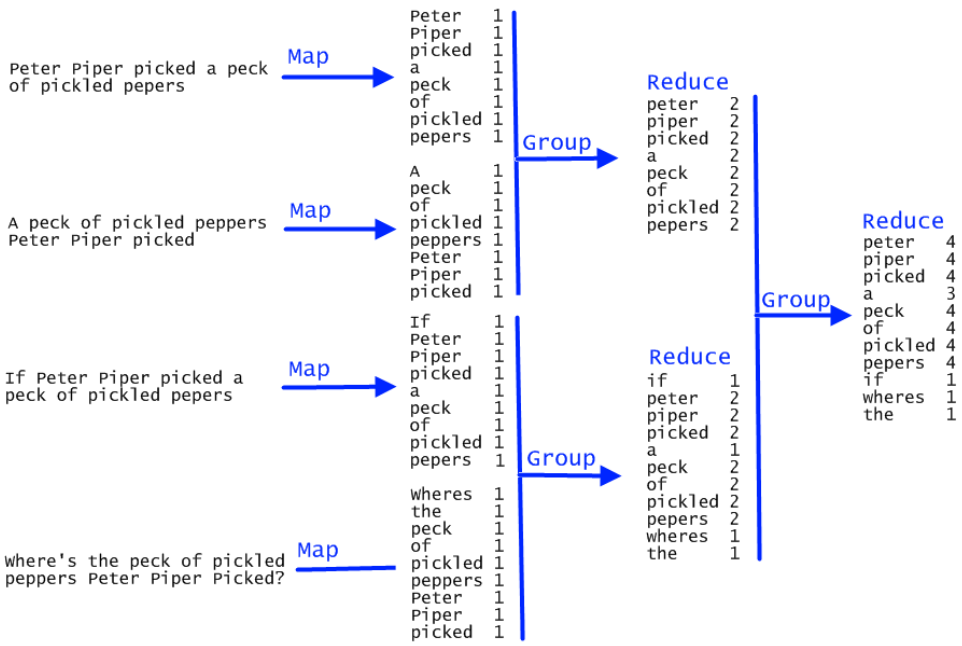

FIG. 2.6: Representação de Map-Reduce.

possa distribuir lotes de entradas para a operação de mapeamento de forma independente, e que as saídas dessas operações possam ser agrupadas para serem tratadas por uma ou mais operações de redução, também de modo independente e paralelo.

2.1.5 SELETIVIDADE

Em (NAVATHE, 2014a) é definido o conceito de **seletividade de um atributo** como sendo a ”fração de registros que satisfazem uma condição de igualdade no atributo”. Seja a tabela representada pela figura 2.7, o valor da seletividade do atributo ”Atributo 1” quando este tem valor igual a ”c”, é igual ao número de vezes que o ”Atributo 1” é igual a ”c” sobre o número total de tuplas da tabela representada pela figura 2.7, ou seja, 0,8.

Atributo 1	Atributo 2
a	1
b	2
c	3
c	4
c	5
c	6
c	7
c	8
c	9
c	10

FIG. 2.7: Tabela exemplo para o cálculo da seletividade.

Há alguns casos onde a quantidade de dados não é suficiente para se calcular a seletividade dos filtros das consultas, ou os dados não existem, ou ainda não foram inseridos em um SGBD relacional. Nestes casos, uma boa estimativa para a seletividade dos filtros é o inverso da cardinalidade do atributo. Por exemplo, se um atributo possui 100 valores distintos, uma boa estimativa para a sua seletividade é 0,01.

Ainda em (NAVATHE, 2014a) é definido o conceito de **seletividade da condição**, como sendo ”a fração de tuplas selecionada por uma condição de seleção”.

Seja a tabela da figura 2.7, a seletividade da condição de seleção expressa na linguagem SQL ”WHERE Atributo 1=”c” and Atributo 2=”3”, equivale ao número de tuplas onde ”Atributo 1” é igual a ”c” e ”Atributo 2” é igual a ”3”, sobre o número de tuplas da tabela. Ou seja, neste caso a seletividade da condição de seleção acima é 0,1.

Seja uma consulta envolvendo as tabelas 1 e 2 apresentadas na figura 2.8, escrita na liguagem SQL da seguinte forma:

```
SELECT * from  Tabela 1 Tabela 2
WHERE Atributo_2 = Atributo_3 and
Atributo_1 = c and Atributo_2 = 3 and
Atributo_4 = "valor3"
```

Tabela 1	
Atributo 1	Atributo 2
a	1
a	4
a	3
b	4
b	1
b	2
c	1
c	2
c	3
c	4

Tabela 2	
Atributo 3	Atributo 4
1	valor1
2	valor2
3	valor3
4	valor3

Tabela 3		
Atributo 1	Atributo 2	Atributo 4
a	1	valor1
a	4	valor3
a	3	valor3
b	4	valor3
b	1	valor1
b	2	valor2
c	1	valor1
c	2	valor2
c	3	valor3
c	4	valor3

FIG. 2.8: Junção de tabelas.

Para uma melhor compreensão do comportamento dos diferentes SGBD em distintas modelagens, foram concebidos dois novos tipos de seletividade, **seletividade de uma tabela na condição** e **seletividade dos campos raízes dos documentos de uma coleção em uma consulta**.

A **seletividade de uma tabela na condição** é equivalente à seletividade da condição levando em consideração somente a condição dos atributos da tabela na qual se deseja saber a seletividade, *após a realização da junção* de todas as tabelas envolvidas na consulta. Ou seja, na consulta acima representada, a seletividade da tabela 1 na condição de seleção é 0,1, pois na tabela 3, que representa o resultado da junção das tabelas 1 e 2, existe somente uma linha na qual ”Atributo 1”=”c” e ”Atributo 2”=”3’, e a seletividade da tabela 2 é 0,2. Note que o Atributo 2 é o mesmo do Atributo 3, por isso foi considerado o Atributo 2 como pertencente às duas tabelas após a junção das mesmas.

Já a **seletividade dos campos raízes dos documentos de uma coleção em uma consulta**, como o próprio nome diz, é a seletividade dos filtros utilizados em uma consulta, de campos que pertencem à raiz dos documentos de uma determinada coleção. Ao realizarmos o seu cálculo, nos bancos de dados de documentos, se o fizermos considerando somente o número de documentos que satisfazem determinada condição dos campos raízes e dividirmos este número pelo número total de documentos, estaremos desconsiderando a quantidade de campos aninhados existentes em cada documento. Entretanto, devido ao fato da operação de desagregação em um número elevado de documentos poder ter um custo elevado para os SGBD que realizam esta operação, o cálculo da seletividade deve considerar o número de documentos aninhados nos agregados.

Para fazer como sugerido acima, é necessário que os documentos sejam desagregados, por meio da operação de desagregação. Ao realizarmos esta operação, o número de documentos existentes será igual ao número de tuplas decorrentes da junção entre as tabelas

utilizadas para construir os agregados de uma coleção. Portanto, o resultado do cálculo da seletividade dos campos raízes de uma determinada coleção em uma consulta será igual ao cálculo da seletividade da tabela utilizada como base para os agregados desta coleção.

Em outras palavras, ao agregar dados de duas tabelas A e B de um esquema relacional, aninhamos uma lista de tuplas de uma tabela B, sob cada tupla relacionada de uma tabela A. Então, ao realizarmos a operação de desagregação, teremos tantos documentos quantas seriam as tuplas em uma junção entre A e B. O parágrafo a seguir apresenta um exemplo mais detalhado deste cálculo.

Suponha que a partir das tabelas 1 e 2 da figura 2.8 fossem construídos agregados cuja modelagem utilize como base a tabela 2. Se desejarmos calcular a seletividade dos campos raízes da coleção que possui esses agregados, na última consulta apresentada anteriormente, o resultado obtido será o mesmo que o valor da seletividade da tabela 2 na condição. Ou seja, o valor da seletividade dos campos raízes dos documentos da coleção que teve a tabela 1 como base para sua modelagem será 0,1 naquela consulta. E a seletividade dos campos raízes dos documentos da coleção que teve a tabela 2 como base para sua modelagem na consulta será 0,2.

Neste trabalho, a partir desta seção, utilizaremos a sigla SCR para nos referir à seletividade dos campos raízes dos documentos de uma coleção em uma consulta.

2.2 SGBD ORIENTADOS A DOCUMENTOS

Segundo Fowler e Sadalage (2013), nos SGBD orientados a documentos o principal conceito são os documentos, podendo ser armazenados neste tipo de SGBD documentos no formato XML (Extensible Markup Language), JSON (JavaScript Object Notation), BSON (Binary JSON) e outros tipos. Estes tipos de documentos são hierárquicos e se comportam como agregados, onde não é necessário haver uma definição de esquema para a inserção de dados.

Neste tipo de SGBD, pode-se afirmar que o metadado das coleções está embutido nos documentos, onde para cada dado há uma descrição do que representa aquele dado. Esta descrição costuma ser denominada chave e o dado em si é chamado de valor. Nos SGBD orientados a documentos é possível realizar consultas no interior dos documentos (navegando em sua hierarquia), bem como realizar consultas mais complexas voltadas para *Online Analytical Processing* (OLAP).

Entre os SGBD orientados a documento mais populares, encontrados no site DB-

engines[7], selecionamos os SGBD MongoDB, ElasticSearch, CouchDB, MarkLogic, OrientDB, pois além de estarem dentre os primeiros no ranking deste site, possuíam algumas características necessárias para este trabalho.

Uma primeira característica para a seleção desses SGBD é a sua capacidade de realizar consultas analíticas, com funções de agregação (SUM, COUNT). Por este motivo, o SGBD DynamoDB que estava bem posicionado no ranking, foi excluído deste estudo. Outro SGBD que encontra-se bem posicionado no ranking, mas que foi eliminado do estudo foi o Couchbase. O Couchbase é um SGBD que utiliza como armazenamento principal a memória principal, e como o presente trabalho teve como foco SGBD que utilizavam a memória secundária, este foi eliminado do estudo.

2.2.1 MONGODB

O MongoDB é um banco de dados de código aberto orientado a documentos que armazena documentos no formato BSON (Binary JSON). O BSON é um formato similar ao formato JSON que permite mais leveza aos dados, maior velocidade e maior facilidade de percorrer caminhos pelos dados que o formato JSON.

Este SGBD armazena os documentos em coleções que podem ser equiparadas a tabelas nos SGBD relacionais[8], mas como já dito anteriormente, os documentos não precisam ter o mesmo esquema. O tamanho máximo de um documento no MongoDB é de 16MB, sendo este, também, o limite para o MongoDB importar arquivos. Ao ser criado um documento, automaticamente é criada uma chave primária para cada documento. Esta chave também pode ser definida previamente, podendo ser composta (com mais de um atributo).

Para inserir documentos no MongoDB, este possui uma ferramenta conhecida como "mongo-import", que permite que documentos de até 16MB sejam inseridos com facilidade.

O MongoDB possui atualmente três mecanismos de gerenciamento dos dados, abaixo relacionados:

Wired Tiger[9]: A partir da versão 3.0 do MongoDB, este vem sendo o mecanismo padrão de gerenciamento dos dados. O Wired Tiger utiliza no mínimo 1GB da memória RAM, ou 60% desta. (HOWS et al., 2015).

MMAPv: É o mecanismo original do MongoDB, deixando de ser o mecanismo padrão a partir da versão 3.2. Este mecanismo é baseado no mapeamento de arquivos de memória,

[7] http://db-engines.com/en/ranking/document+store
[8] mongodb.org
[9] https://docs.mongodb.com/manual/core/wiredtiger

possuindo uma boa performance em carregamentos de grande quantidade de leitura e escrita, porém segundo Hows et al. (2015) este possui menor performance que o WiredTiger.

Mecanismo de alocação em memória: Este mecanismo somente está disponível na versão Enterprise, sendo que este mecanismo ao invés de alocar os dados no HD, os aloca na memória principal.

O principal local de armazenamento dos dados no MongoDB é o disco rígido onde este utiliza tanto a cache do *Wired Tiger*, quanto a cache do sistema de arquivos do sistema operacional como espaço de memória RAM disponível. No sistema operacional LINUX a política de acesso à memoria RAM, para o MongoDB é o *Less Recently Used* (LRU).

O MongoDB possui uma alta flexibilidade no seu esquema, isto é, a maneira que os dados são organizados é reconhecida automaticamente pelo SGBD a partir da estrutura do documento, não sendo necessário definir previamente qualquer tipo de esquema para o banco. Entretanto, para a realização de consultas é desejável que, em cada coleção, os documentos possuam a mesma estrutura.

Existem duas formas de se realizar consultas no MongoDB, onde o usuário pode utilizar a função Map-Reduce da linguagem Javascript ou uma linguagem própria deste SGBD conhecida como *Aggregate-Framework*. Neste trabalho somente serão avaliadas as consultas realizadas com o *Aggregate-Framework* devido ao seu melhor desempenho, segundo Chodorow (2013). Com relação à facilidade de compreensão da linguagem do *Aggregate-Framework*, pode-se afirmar que o MongoDB possui uma linguagem de nível médio. As operações nesta linguagem são definidas uma após a outra, como em um pipeline, permitindo que a maioria das operações realizadas pela linguagem SQL sejam realizadas de forma clara. Na seção 4.3 serão apresentados exemplos de uso desta linguagem.

Os tipos de índices possíveis de se utilizar no MongoDB são índices de *hash*, índices de ordenação de *arrays* e de procura de textos, onde este utilizam uma estrutura de dados de árvores-B. O MongoDB permite a criação de índices em campos isolados, índices compostos e índices parciais, porém existem algumas restrições para a criação de índices compostos. A criação de índices compostos somente é possível quando este não envolve campos da estrutura dos documentos que são listas e que estão em ramos paralelos.

Por exemplo, o índice composto definido por (id, vendas.data.dia, vendas.item.valor) não poderia ser criado pois ambos os campos *data* e *item* são listas que estão em ramos paralelos da hierarquia. Já o índice composto (id, vendas.data.dia, vendas.data.mes) poderia ser criado, pois apesar de os atributos *dia* e *mês* serem atributos "irmãos", estes não são listas.

O MongoDB utiliza a interseção de índices de campos individuais, ou de campos individuais e parte dos índices compostos para otimizar ainda mais a consulta, porém esta interseção somente ocorre em dois campos[10].

Havendo a possibilidade de utilizar mais de um índice em uma consulta, o MongoDB possui um algoritmo próprio que calcula qual índice será utilizado de forma a obter a melhor performance. Porém, se o usuário desejar, é possível ainda definir qual índice o MongoDB deve utilizar. Para forçar a utilização de índices, é necessário alterar a expressão da consulta do MongoDB (*hints*).

2.2.2 ELASTICSEARCH

O ElasticSearch (ES) é um mecanismo de busca de dados baseado no Apache Lucene que permite distribuir dados e escalonar dados, assim como um banco de dados de documentos, sendo que alguns sites[11] o classificam como um banco de dados orientado a documentos. Existem algumas discussões a respeito da classificação do ES como um banco de dados NoSQL, porém pode-se dizer que a sua classificação é tão inexata quanto à definição do termo NoSQL (BRASETVIK, 2013).

O ES utiliza como padrão 1GB de memória heap da Máquina Virtual do Java (JVM), podendo ser aumentada até 32GB. Em alguns tipos de consultas é necessário trazer certos campos de todos os documentos para a memória o que pode causar falta de memória nos nós e, consequentemente, a queda do nó.

Ao se inserir um documento no ES é criado um índice para cada campo do documento. Além disso, é definido o tipo de dados básico daquele campo (i.e., inteiro, alfanumérico, etc...), de modo automático.

Para que seja possível expressar consultas sobre os campos aninhados dos documentos, é necessário definir um esquema, o que faz com que o ES deixe de ser flexível (KONONENKO et al., 2014). Assim, quando o documento possui campos aninhados, conhecidos como NESTED no ES, ao definir o esquema, é preciso indicar quais são estes campos antes de inserir os documentos. Para cada documento aninhado será criado um novo documento, e é feito um mapeamento entre o documento aninhado e o documento "pai". Esta forma de organizar os dados aninhados faz com que não seja necessário realizar uma operação de desagregação, explicada anteriormente, nas consultas realizadas no ES.

Existe ainda outra forma de organizar os dados no ES, quando existem documentos com campos aninhados, conhecida como *parent-child*. Mesmo sendo esta forma dita mais

[10] https://docs.mongodb.com/v3.2/core/index-intersection/
[11] nosql-database.org

flexível que a descrita anteriormente, neste trabalho, esta forma não será avaliada devido ao fato de que as consultas sobre documentos organizados dessa forma podem ser de 5 a 10 vezes mais lentas[12]. Além disso, esta forma não tem suporte para operações de agregação em campos da raiz dos documentos, o que impossibilita a realização de algumas consultas selecionadas para este trabalho.

O ES possui duas formas de realizar consultas. A forma mais simples é denominada *lite* e consiste em chamadas utilizando o método GET, onde os parâmetros são passados no próprio método. Porém, esta forma de realizar consultas é limitada, não sendo possível realizar consultas analíticas.

A outra forma de realizar consultas no ES utiliza uma linguagem de consulta conhecida como Query DSL, sendo esta uma linguagem de difícil entendimento se comparada com a linguagem SQL. Esta linguagem possui uma restrição quando é necessária a ordenação de resultados em operações de agregação. Não é possível realizar operações de ordenação de mais de um campo quando o primeiro campo com o qual se deseja ordenar pertence aos documentos aninhados, e o segundo campo é um campo da raiz dos documentos. Devido a esta limitação do ElasticSearch, visando possibilitar que as consultas em todos os SGBD estudados fossem equivalentes, as operações de ordenação não fizeram parte deste estudo. Exemplos desta restrição serão apresentados na seção 4.3.

Na Query DSL, existem duas maneiras distintas de realizar busca por valores. Na primeira, a busca é feita para valores exatos, atendendo aos filtros definidos. A outra maneira retorna documentos que possuem valores aproximados. A busca por valores aproximados é mais utilizada quando existem campos que contêm textos de mais de uma palavra, podendo, por exemplo, retornar documentos sem fazer distinção entre letras maiúsculas ou minúsculas (GORMLEY; TONG, 2015).

A principal diferença entre esses dois modos de consulta está relacionada com o fato de que a busca por palavras exatas retorna uma simples lista de documentos, conhecida como *bitset*, que satisfazem a condição de seleção da consulta, que por ser de menor tamanho, esta lista pode permanecer na memória principal por mais tempo, agilizando as consultas subsequentes que utilizem o mesmos filtros, o que não acontece quando a maneira de realizar a consulta utiliza buscas aproximadas. Porém, neste modo, é criado um índice invertido para cada palavra, o que corresponde a um índice de *bitmap* para cada palavra isolada do texto (CHAN; IOANNIDIS, 1998), permitindo agilizar as consultas. Neste trabalho, para as consultas dos experimentos definidos, somente o primeiro modo foi usado por ser o modo mais eficiente.

[12](https://www.elastic.co/guide/en/elasticsearch/guide/current/parent-child-performance.html)

Uma diferença encontrada no ES, em relação aos demais SGBD considerados neste estudo, é que ao realizar operações de agrupamento, os campos com valores nulos não são agrupados. Por exemplo, se desejamos agrupar por idade e nome, mas há valores nulos para a idade em alguns documentos, o ES não realiza o agrupamento para esses documentos com valores nulos, não aparecendo o grupamento nulo no resultado.

2.2.3 ORIENTDB

O OrientDB é um SGBD orientado a documentos e também orientado a grafo escrito em JAVA, que possui a capacidade de armazenar e gerenciar tanto grafos quanto documentos. Porém, os relacionamentos nos documentos são gerenciados como em um banco de dados orientado a grafos (ABUBAKAR et al., 2014). Neste trabalho o OrientDB somente será avaliado na categoria de SGBD orientado a documentos.

No OrientDB existem duas formas de organizar um documento no formato JSON, a primeira forma consiste em tratar o documento todo como uma única classe, onde dentro da única classe de documentos criada estarão os documentos com todos os seu campos aninhados. Esta forma de organizar os dados no OrientDB neste trabalho será intitulada documentos aninhados.

A segunda forma de organizar os dados no OrientDB consiste em separar os campos aninhados em classes diferentes, onde para cada tipo de campo aninhado é criada uma nova classe e o relacionamento entre os dados das raízes dos documentos e os dados dos campos aninhados são mantidos por meio de mapeamento. Esta forma de organizar os dados é conhecida como documentos ligados (GARULLI, 2013).

Este trabalho estuda o impacto da modelagem de agregados em ambas as formas de organizar os dados de forma a possibilitar uma análise do desempenho em ambas as formas.

O OrientDB possui uma ferramenta de exportação e importação de dados chamada ETL. Para utilizar esta ferramenta é necessário definir as classes que estão relacionadas nos documentos. Logo, para usar esta ferramenta o esquema de dados deve ser definido.

É possível ainda inserir dados neste SGBD sem utilizar a ferramenta ETL através de um comando *INSERT* nas aplicações clientes. Porém, para o OrientDB utilizando documentos ligados, é necessário definir dentro da estrutura do documento a classe a que pertencem os campos aninhados, o que obriga o usuário a conhecer a estrutura do seu documento e ter que inserir informações do esquema dos documentos a serem inseridos. Se o usuário escolher utilizar o OrientDB sem ligação entre as classes, estes podem ser

inseridos somente em uma classe, sendo que neste caso, os dados aninhados serão tratados como *EMBEDDED*, não havendo necessidade de se definir qualquer esquema na estrutura dos documentos a serem inseridos.

Há pouca documentação do OrientDB para utilizá-lo como um SGBD orientado a documento. A maior parte da documentação é voltada para a sua utilização como orientado a grafo.

O OrientDB possui suporte a 4 tipos de mecanismos de gerenciamento de memória, a saber (GARULLI, 2013):

- plocal: Utiliza armazenamento em disco e trabalha com modelo de paginação. Os componentes do mecanismo plocal acessam o conteúdo do disco através da memória cache.

- remoto: Utiliza a rede para acessar os dados armazenados remotamente.

- memória: Todo o dado permanece na memória principal.

- local: Este tipo de gerenciamento de memória foi substituído pelo plocal e utilizava a segmentação de dados segundo os clusters de classes criadas.

A linguagem de consulta para documentos utilizada pelo OrientDB é de fácil compreensão devido à sua similaridade com o SQL.

Para cada documento inserido no OrientDB é criada uma chave que identifica individualmente cada documento e um índice para esta chave. Havendo documentos ligados, para cada documento ligado também é criada uma chave e um índice para a chave.

O OrientDB possui três algoritmos de indexação, o SB-Tree, Hash-Index e o Lucene-Engine, sendo que os dois primeiros permitem a criação de índices únicos, não únicos, índices de palavras e de dicionário. Este último é um índice equivalente ao índice único, porém cria um novo índice caso haja entradas duplicadas. O algoritmo de indexação Lucene-Engine permite ainda a criação de índices sobre textos maiores que uma palavra e sobre coordenadas geográficas. O algoritmo padrão utilizado pelo OrientDB é o SB-Tree (GARULLI, 2013).

A criação de índices no OrientDB utilizando a organização de documentos aninhados no OrientDB somente é possível em campos das raízes dos documentos, não sendo possível criar índices em campos aninhados.

Utilizando documentos ligados no OrientDB apesar de ser possível criar índices em campos aninhados, o OrientDB somente reconhece aqueles existentes na classe onde a consulta foi realizada. Portanto, o OrientDB somente reconhece os índices da classe raiz dos documentos.

No OrientDB, tanto utilizando documentos aninhados como documentos ligados, é possível a criação de índices compostos nos campos das raízes dos documentos, porém

para utilizá-los é necessário especificar o índice que está sendo utilizado, pois para índices compostos o OrientDB trata os índices compostos como um único campo de cadeia de caracteres. Para índices sobre campos individuais não é necessário realizar qualquer tipo de alteração nas consultas, pois se a consulta pode utilizar um índice, o OrientDB irá utilizá-lo automaticamente[13].

2.2.4 COUCHDB

O CouchDB é um SGBD orientado a documentos que arquiva os dados no formato JSON (FOUNDATION, 2015), possuindo uma API que permite acessar os dados por um web *browser* via HTTP, de modo a servir principalmente para aplicações que utilizam servidores web e para aplicativos de celulares.

Os dados do CouchDB são armazenados no disco rígido do computador e são acessados somente para atualizações ou remoções, onde o método de acesso ao banco de dados pode ser por meio de comandos POST, GET, PUT e DELETE (PHAN et al., 2014) ou por meio da API FUTON, que é a API em HTTP do CouchDB.

Assim como o MongoDB, o CouchDB permite que listas de documentos sejam importados diretamente para o CouchDB sem que seja necessário definir o esquema dos dados. No CouchDB um arquivo pode ter um tamanho de até 4GB.

Para realizar consultas no CouchDB é necessário criar uma visão materializada a partir de uma função Map-Reduce, na linguagem Javascript, permitindo assim que as consultas sejam realizadas sobre esta através de requisições HTTP, utilizando o método GET.

Algumas restrições impedem que certos tipos de consultas sejam realizadas[14]. O principal motivo é que as consultas são feitas sobre as chaves geradas e estas chaves são tratadas como uma cadeia de caracteres. Por exemplo, seja uma visão materializada gerada, cuja chave é composta pelos atributos nome, cidade, país e idade e o atributo salário como um atributo valor (obtido pelo agrupamento nos campos nome, cidade, país e idade). Nesta visão é possível realizar uma consulta com o filtro nome = ”Pedro”, agrupando por cidade, país e idade. Porém, não seria possível agrupar somente por idade a partir desta visão materializada. Portanto, se desejarmos realizar esta a consulta seria necessário criar outra visão materializada para tal.

[13]http://orientdb.com/docs/2.1/SQL.html
[14]http://docs.couchbase.com/admin/admin/Views/views-querySample.html

2.2.5 MARKLOGIC

O MarkLogic é um SGBD NoSQL que combina a estrutura interna de um banco de dados, com um estilo de busca baseado em índices e o comportamento de um servidor de aplicação em um sistema unificado (HUNTER; WOOLDRIDGE, 2016). Atualmente o MarkLogic é categorizado como um SGBD orientado a documentos, nativo em documentos XML, podendo armazenar arquivos em RDF e ainda ser um mecanismo de busca. Por isso, pode ser classificado como um SGBD multi-modo. Segundo Hunter e Wooldridge (2016), este sistema possui as características ACID dos bancos de dados transacionais.

Apesar de possuir as características multi-modais, este SGBD somente é avaliado neste trabalho com relação às suas capacidades de atuar como um banco de dados orientado a documentos.

O MarkLogic, além de realizar a leitura de arquivos em XML e em RDF, também tem suporte para arquivos em JSON, bem como documentos em texto e documentos em formato binário, não havendo necessidade de definição de esquemas para nenhum dos tipos de arquivos.

O MarkLogic possui atualmente 4 linguagens de consulta, XQuery, XSLT, Javascript e SPARQL. É possível misturar as linguagens para realização de consultas, isto é, uma expressão em uma linguagem pode utilizar algumas funções das outras linguagens.

O MarkLogic aceita requisições em HTTP, podendo executar chamadas utilizando XQuery, XSLT ou Javascript, permitindo a criação de páginas dinâmicas, bem com a sua utilização como uma REST API.

No MarkLogic cada banco de dados contém coleções que são chamadas de florestas, podendo haver mais de uma floresta em cada banco de dados. Para cada floresta é criado um diretório no disco, onde são armazenados os documentos, os índices e o *journal* daquele banco de dados. Cada floresta é dividida ainda em armazéns, chamados de *stands*, sendo que, por ocasião da inserção dos documentos, parte dos armazéns são mantidos na memória principal até que estes alcancem o seu limite. À medida que a quantidade de dados aumenta, os *stands* são mesclados entre si, pois cada um armazena os índices dos seus documentos, de forma a possibilitar a criação de um índice único para toda a floresta.

Mesmo sem ter conhecimento do esquema, o MarkLogic indexa os documentos em XML ou em JSON, por ocasião da sua inserção, o que permite uma enorme flexibilidade e a manutenção do desempenho em consultas mesmo após a inserção de novos campos, pois estes serão indexados automaticamente. Porém, os índices criados automaticamente pelo MarkLogic são índices que agilizam as consultas sobre valores exatos. Portanto, para

otimizar consultas que utilizam filtros com intervalos de valores, é necessária a criação dos índices de intervalo, sendo que para sua utilização, a consulta deve ser reescrita de forma a sinalizar que estes índices devem ser usados.

Além desses dois tipos de índices, existem outros tipos que podem ser criados, como por exemplo, índices de busca de frases com mais de uma palavra (HUNTER; WOOLDRIDGE, 2016), porém para a realização das consultas neste trabalho, além dos índices criados automaticamente, somente foram utilizados os índices de intervalo.

O MarkLogic não possui limitações quanto à criação de índices em campos aninhados e a interseção desses ocorre entre um número ilimitado de índices. Entretanto, no MarkLogic não é possível a criação de índices compostos[15].

2.2.6 COMPARAÇÃO DOS SGBD ORIENTADOS A DOCUMENTOS

Na tabela 2.1 podem ser visualizadas algumas comparações das principais características dos SGBD estudados neste trabalho que foram avaliadas por este autor:

TAB. 2.1: Resumo das principais características dos SGBD.

SGBD	Entendimento da linguagem	Flexibilidade de Esquema	Índices automáticos	Índice em campos aninhados
MongoDB	Médio	Alta	Não	Sim
Elasticsearch	Difícil	Média	Sim	Sim
OrientDB	Fácil	Média	Não	Não
CouchDB	Médio	Alta	Não	Não se aplica
MarkLogic	Médio	Alta	Sim	Sim

Na segunda coluna da tabela, são apresentados os níveis de dificuldade de compreensão da linguagem de consulta, de acordo com a experiência do autor no decorrer deste trabalho.

Devido ao OrientDB ter uma linguagem similar ao SQL, a sua linguagem de consulta pode ser considerada de fácil compreensão. Já o MarkLogic utiliza a linguagem XQuery 1.0, que por ser um pouco mais difícil que o SQL, foi considerada como uma linguagem de nível médio. O CouchDB utiliza como linguagem para construção das visões materializadas a função Map-Reduce em Javascript, que é uma função muito similar ao XQuery, por isso pode ser considerada também de nível médio. Como a linguagem Aggregate Framework do MongoDB, realiza operações sequenciais, a sua compreensão não é dificultada, podendo ser considerada também de média compreensão. Já a linguagem

[15]https://developer.marklogic.com/blog/how-is-marklogic-different-from-mongodb

Query-DSL do ElasticSearch, utilizada para consultas analíticas, possui uma linguagem de difícil compreensão, principalmente com relação às funções de agrupamento.

Quanto à flexibilidade de esquema, o MarkLogic, o CouchDB e o MongoDB não necessitam da definição de nenhum esquema para a realização de quaisquer operações, podendo ser considerados bancos com alta flexibilidade de esquema. O OrientDB, utilizando documentos aninhados pode ser considerado com alta flexibilidade, porém para a utilização de índices é necessário criar as propriedades na qual se deseja utilizar os índices, perdendo um pouco de flexibilidade. Já para documentos ligados, como é necessário definir nos documentos em JSON, qual a classe a que pertencem os dados, este pode ser considerado um SGBD com média flexibilidade de esquema. Já o ElasticSearch, devido à necessidade de se definir campos aninhados, para que estes possam ser consultados, pode ser considerado como de média flexibilidade de esquema.

Entre os SGBD estudados somente o MarkLogic e o ElasticSearch criam índices automaticamente e dentre todos eles, o único que não permite utilizar índices em campos aninhados é o OrientDB. O CouchDB, por utilizar como padrão as consultas às visões materializadas, não necessita criar índices para os campos aninhados, pois os índices são criados para as visões materializadas.

2.3 BENCHMARKS

Nesta seção serão apresentados de forma resumida os *benchmarks* utilizados para os experimentos deste estudo, o *Star Schema Benchmark* e o Transaction Processing Performance Council - Decision Suport (TPC-DS).

Esses benchmarks foram escolhidos em virtude de terem sido criados com o intuito de avaliar o desempenho de sistemas de apoio à decisão. Ambos os *benchmarks* permitem a geração de dados em escalas variadas e possuem um conjunto de consultas analíticas pré-definidas que simulam uma situação próxima à realidade de sistemas de apoio à decisão.

Portanto, o estudo do impacto da modelagem de agregados a partir desses *benchmarks* poderá ser utilizado como referência para sistemas que necessitem de ferramentas de suporte à decisão, que utilizam, ou têm a intenção de utilizar, um dos SGBD aqui estudados.

2.3.1 STAR SCHEMA BENCHMARK

Segundo O'Neil et al. (2009), o Star Schema Benchmark (SSB) foi desenvolvido para avaliar a performance de bancos de dados no apoio a aplicações de data warehouse. O

SSB é baseado no Benchmark TPC-H, com algumas modificações. A idéia principal deste *benchmark* é padronizar a organização dos dados e estabelecer as consultas que devem ser realizadas sobre os dados para avaliar os diferentes SGBD. Inicialmente este *benchmark* foi criado para avaliar SGBD relacionais, porém alguns trabalhos como o de Carniel et al. (2012b) demostraram a possibilidade de utilizar o SSB para avaliar os SGBD NoSQL.

Este *benchmark* tem um esquema com cinco tabelas, sendo uma tabela de fatos e quatro de dimensão, conforme a figura 2.9. Isso possibilitou a variação da modelagem dos agregados em pelo menos cinco formas diferentes, onde os dados podem, ser aninhados em até três níveis. Para o esquema citado, o *benchmark* provê um conjunto de consultas analíticas pré-estabelecidas que podem ser utilizadas para a análise do comportamento dos SGBD estudados.

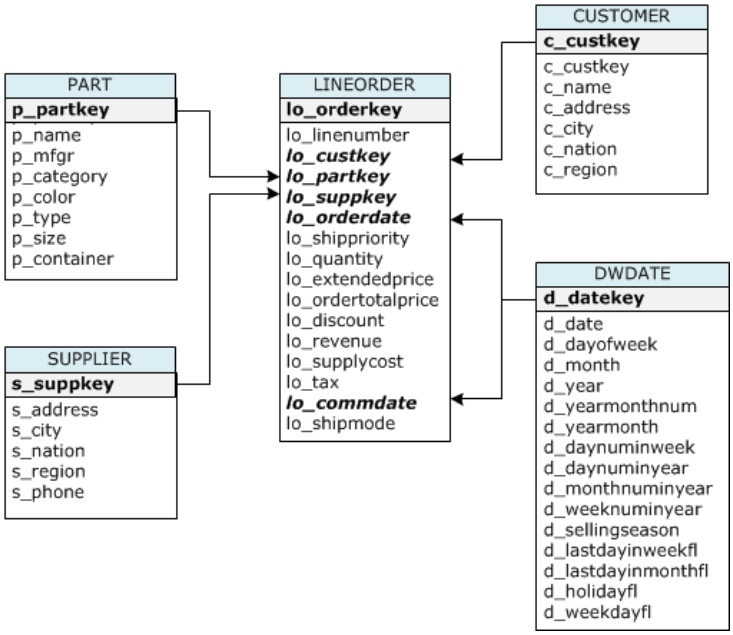

FIG. 2.9: Star Schema Benchmark.

No endereço eletrônico do SSB[16], é possível obter um gerador de dados para utilizar nos SGBD avaliados, podendo ser criados diferentes quantidades de dados para diversas cargas nos bancos de dados, onde o padrão utilizado é o fator de escala 1 que irá criar 1 GB de dados. Neste fator são criados 6 milhões de tuplas para a tabela *Lineorder*, 200 mil para a tabela *Part*, 30 mil para a tabela *Customer*, 2 mil para a tabela *Supplier* e sete anos de datas para a tabela *Date*.

[16]https://github.com/electrum/ssb-dbgen

2.3.2 TPC-DS BENCHMARK

O TPC-DS é um *benchmark* criado com a finalidade de avaliar a performance de sistemas de apoio à decisão que examinam grande volumes de dados e respondem às consultas relacionadas com negócios do mundo real. Este *benchmark* possui um total de sete esquemas de dados, onde seis desses esquemas possuem o objetivo de contabilizar os resultados de vendas e retorno de três canais de venda: internet, catálogo e lojas. O sétimo esquema consiste em um controle de inventário dos três canais de venda. A figura 2.10 representa um dos esquemas do TPC-DS, o esquema *Store_Sales*.

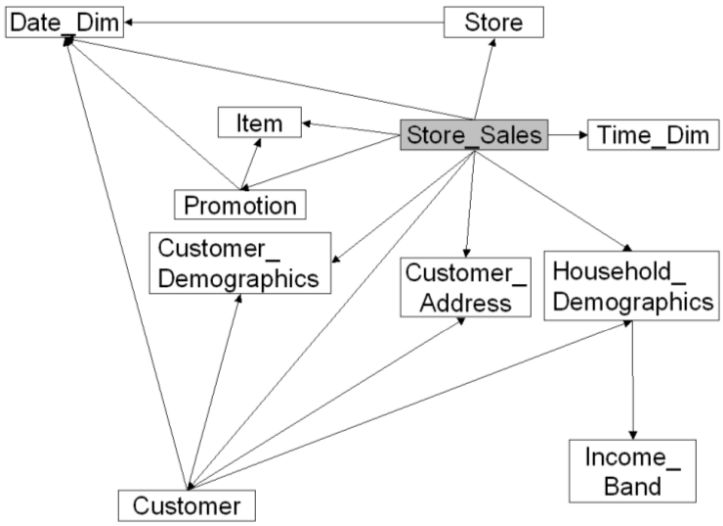

FIG. 2.10: Esquema Store Sales original do TPC-DS.

O TPC-DS abstrai a diversidade de operações encontradas usualmente em aplicações de análise de informação, mantendo as características essenciais de desempenho. Como é necessário executar um grande número de consultas e transformações de dados para gerir completamente qualquer ambiente de análise de negócios, o TPC-DS define 99 consultas distintas e 12 operações de manutenção de dados, cobrindo as operações típicas de um sistema de suporte à decisão, como consultas *ad-hoc*, relatórios iterativos (*drill down / drill up*) e consultas de extração e atualização periódica da base de dados[17].

A relação deste trabalho com este *benchmark* limitou-se à utilização dos dados gerados pelo seu gerador de dados e à realização de parte das consultas pré-estabelecidas pelo TPC-DS, visando validar as heurísticas que serão apresentadas mais adiante. Portanto, não é finalidade deste trabalho realizar o *benchmark* completo, pois o objetivo do TPC-DS não é o mesmo do presente estudo.

[17]https://www.cisuc.uc.pt/ckfinder/userfiles/files/08%20-%20Pedro%20Furtado%20(2).pdf

A escolha deste *benchmark* para a validação das heurísticas deve-se ao fato deste possuir um conjunto de dados similares aos do SSB, com um conjunto de consultas com objetivos parecidos. Entretanto para a sua utilização foram necessárias algumas modificações que serão explicadas no início do capítulo 5.

3 TRABALHOS RELACIONADOS

À medida que a importância dos bancos de dados NoSQL vem crescendo no atual contexto de computação em nuvem, Internet of Things e no mundo Big Data, vem aumentando o número de estudos abordando os bancos de dados NoSQL. Os trabalhos relacionados levantados se resumiram aos que tratavam a respeito da modelagem de dados em bancos de dados NoSQL e aos que realizavam a comparação de desempenho entre os diversos sistemas de banco de dados SQL e NoSQL, em virtude destes dois tipos estarem diretamente ligados aos objetivos deste trabalho.

3.1 TRABALHOS DE MODELAGEM DE BANCOS DE DADOS NOSQL, XML E ORIENTADOS A OBJETO

Existem atualmente poucos trabalhos que abordam a modelagem de dados nos bancos de dados NoSQL, que são discutidos nesta seção. A maioria tem foco na conversão de modelos conceituais para o modelo lógico em bancos de dados NoSQL. Além disso, antes do surgimento dos bancos de dados NoSQL, existiam dois modelos de bancos de dados que possuíam algumas características em comum com os bancos de dados NoSQL, os bancos de dados XML e os Orientados a Objeto, pois são bancos que possuem dados que podem ser hierarquizados em estruturas aninhadas e utilizar campos multivalorados. Alguns trabalhos que abordaram estes modelos também são discutidos brevemente nesta seção.

No trabalho de Hoberman (2014), é discutida a modelagem dos dados no SGBD MongoDB desde a modelagem conceitual até a modelagem física dos dados. Neste trabalho a modelagem dos dados é vista sob dois enfoques, um voltado para consultas analíticas e outro para operações OLTP.

No final deste trabalho são apresentados cinco aspectos a serem considerados na escolha entre utilizar referências ou aninhar os dados que foram observados no trabalho atual. Estes aspectos sugerem que as modelagens geradas, tanto para o conjunto de dados do SSB, quanto para o TPC-DS, busquem aninhar os dados de modo que as consultas analíticas possam ser melhor atendidas. Porém, o trabalho de Hoberman (2014) não discute a possibilidade de variar a modelagem dos agregados, de acordo com as consultas a serem realizadas e também não realiza experimentos, mostrando o desempenho do MongoDB a

partir das modelagens propostas em seu trabalho.

No trabalho de Fowler e Sadalage (2013), são discutidos diversos aspectos de modelagem de dados em todos os modelos de banco de dados NoSQL, onde é levantada a importância da modelagem dos dados voltada para o acesso aos dados e como os agregados podem ser modelados de forma que não haja necessidade de usar referências entre agregados. Porém, diferente de Hoberman, neste trabalho, além das consequências da orientação a agregados nos sistemas de banco de dados NSQL, é discutido como uma estrutura de um agregado pode ser útil em algumas interações, mas ser um obstáculo para outras, mostrando que a maneira que um agregado é modelado pode influenciar nas operações a serem realizadas, porém este trabalho não discute, nem demonstra, o quanto uma maneira de modelar pode influenciar nas consultas analíticas.

Já o trabalho de Kaur e Rani (2013) propõe uma forma de se obter a modelagem lógica de dois modelos de SGBD NoSQL a partir da modelagem conceitual, tomando como base um modelo ER. Este trabalho utilizou um estudo de caso como exemplo do seu estudo. Diferente do discutido em (HOBERMAN, 2014) e (FOWLER; SADALAGE, 2013) este trabalho, apesar de apresentar um forma de obter a modelagem lógica dos dois modelos, não discute os aspectos que devem ser considerados para a modelagem dos dados, com por exemplo se as entidades devem ser aninhadas ou referenciadas. Além disso, neste trabalho, não é explorada a possibilidade de utilizar diferentes modelagens de agregados, bem como não são apresentados resultados que demonstrem o desempenho dos sistemas nas modelagens propostas.

Diferentemente dos trabalhos anteriores, o trabalho de Chevalier et al. (2015) investiga o uso de dois modelos de SGBD NoSQL somente para sistemas de suporte à decisão. Neste trabalho são apresentadas regras de mapeamento de esquemas conceituais multidimensionais, utilizados em *data warehouse*, para o modelo lógico de bancos de dados NoSQL orientados a coluna e a documentos.

De forma distinta do presente trabalho, em (CHEVALIER et al., 2015), no mapeamento do modelo conceitual para o lógico dos SGBD orientados a documentos, somente foi considerada uma única forma de modelar o agregado, não levando em consideração outras formas. Neste trabalho, vale ressaltar ainda que são apresentadas formas de mapear estruturas de valores pré-computadas para os dois modelos estudados (*lattice* de visões materializadas).

Os experimentos realizados neste trabalho, resumiram-se a investigar o tempo de carga dos agregados formados pelas regras de mapeamento propostas e pela medição do tempo para a geração das estruturas pré-computadas. Foram computados valores de

soma, conta, máximo e mínimo de um atributo da tabela de fato, a partir da combinação de atributos das tabelas de dimensão, utilizando para isso dados de um dos esquemas do TPC-DS, o esquema *Store_Sales*.

Em (SCABORA et al., 2016) é mostrado como a modelagem dos dados em um SGBD orientados a colunas pode influenciar no desempenho de consultas analíticas. Nos SGBD orientados a colunas uma família de colunas é formada por dados que são acessados ao mesmo tempo (FOWLER; SADALAGE, 2013), de forma similar aos agregados nos SGBD orientados a documentos. Neste trabalho foram utilizadas diferentes configurações de famílias de colunas e foram realizados experimentos de forma a mensurar qual modelagem permite um melhor desempenho em consultas analíticas. Os seus resultados mostraram que diferentes configurações beneficiaram diferentes consultas, de acordo com o número de entidades envolvidas nas consultas. Portanto, este trabalho possui objetivos similares ao atual, porém seu estudo ocorre em uma abordagem de SGBD diferente.

O trabalho de Elmasri et al. (2005), por ser mais antigo que os primeiros, aborda uma metodologia de modelagem de esquemas XML a partir da modelagem conceitual. É proposto um algoritmo para a criação de esquemas hierárquicos a partir de esquemas Entidade-Relacionamento Estendidos (EER). Além disso, este trabalho ilustra o processo de geração de consultas SQL que formarão os documentos em XML. A semelhança do trabalho de Elmasri et al. (2005) com o presente trabalho está na ideia de que a partir de um modelo conceitual, é possível criar diferentes visões hierárquicas dos documentos, dependendo da entidade que for selecionada como raiz dos documentos. Embora baseado na mesma ideia, aquele trabalho não estabelece um critério para a escolha de uma visão hierárquica, nem realiza experimentos em SGBD XML.

De forma análoga ao trabalho anterior, em (SCHROEDER; MELLO, 2008) é proposta uma abordagem de modelagem de esquemas XML, partindo da análise do modelo conceitual e do acesso aos dados, por ocasião das operações realizadas nos bancos de dados. Esse trabalho propõe um conjunto de regras a ser seguidas para a criação do esquema XML de modo a obter um esquema que utilize um número menor de referências para o acesso a dados de outras entidades. Esse trabalho mostrou que uma melhor organização da estrutura hierárquica pode beneficiar as consultas realizadas nos bancos de dados XML, ideia esta que é utilizada no presente trabalho em virtude de os dados nos SGBD orientados a documentos também poderem ser organizados em estruturas hierárquicas.

Além desses trabalhos, existem ainda dois trabalhos que discutem a modelagem de dados em SGBD Orientados a Objetos, (NAVATHE, 2014a) e (FONG, 1995). Em ambos os trabalhos são apresentadas regras de mapeamento de esquemas EER para a modelo

lógico deste tipo de SGBD. Porém, o primeiro trabalho é mais detalhado, pois discute aspectos que vão além da modelagem lógica, como por exemplo a escolha entre criar uma classe (com dados aninhados) ou deixar um atributo como sendo uma referência para relacionamentos com outra classe. Em ambos os trabalhos, apesar de discutirem aspectos relevantes para este tipo de SGBD, não foi mensurado o quanto uma modelagem pode beneficiar mais as consultas do que outra modelagem, nem foram criadas heurísticas para auxiliar a escolha de modelagens.

Em resumo, apesar de já existirem alguns trabalhos que apresentam orientações de como a modelagem de dados nos SGDB NoSQL deve ser realizada, ainda está ausente na literatura, estudos que expõem resultados que mostram o impacto da modelagem dos dados, seja em consultas ou em outras operações. Além disso, este trabalho se diferencia dos demais por focar na modelagem do agregado e no impacto da variação da sua modelagem no desempenho de consultas nos SGBD orientados a documentos.

3.2 TRABALHOS DE COMPARAÇÃO DE DESEMPENHO

Existem na literatura diversos trabalhos comparando o desempenho de SGBD, sendo que a maioria dos trabalhos são benchmarks entre os diversos SGBD, comparando o tempo de resposta de consultas, espaço de armazenamento, tempo de carregamento do dados, throughtput, etc. Há entre os trabalhos estudados, alguns que se assemelham em algum aspecto com este trabalho. Os trabalhos de comparação de desempenho foram divididos em duas categorias, comparações entre SGBD SQL e NoSQL e somente entre SGBD NoSQL, e são discutidos a seguir.

3.2.1 SQL X NOSQL

Em (CARNIEL et al., 2012b) foi realizada a comparação de desempenho entre diferentes SGBD, utilizando consultas OLAP (Online Analytical Processing) sobre um Data Warehouse, utilizando para isso as consultas e o esquema do SSB. Foram comparados neste trabalho os seguintes SGDB: MongoDB do modelo de documentos, FastBit e LucidDB do modelo de família de colunas e o PostGreSQL do modelo relacional.

Para os testes foram utilizadas duas bases de dados, geradas utilizando o SSB, uma usando o fator de escala 1, produzindo 6 milhões de tuplas e a outra com o fator de escala 10, produzindo um *dataset* dez vezes maior.

Este trabalho buscou avaliar o desempenho dos SGBD tanto em consultas normais nos bancos, quanto sobre visões fragmentadas verticalmente e visões materializadas, bem

como avaliou o desempenho utilizando índices, permitindo que fossem feitas comparações entre o espaço utilizado para as visões e o desempenho dos SGBD estudados, utilizando as diferentes técnicas de otimização.

Os resultados deste trabalho mostraram que o desempenho do MongoDB foi o pior em todas as configurações e o FastBit utilizando Visões Fragmentadas apresentou os melhores resultados, onde o principal fator desta melhor performance foi a utilização de índices de bitmap de junção.

Este trabalho é diferente do atual em virtude dos testes terem sido realizados somente em uma modelagem de agregado, descartando a possibilidade de as implementações poderem ter melhores desempenhos em outras modelagens.

Já o trabalho de Phan et al. (2014), além de apresentar as principais características que diferenciam os bancos de dados relacionais dos NoSQL, apresenta características detalhadas dos diferentes modelos de dados NoSQL bem como dos SGBD comparados (MySQL, CouchDB, MongoDB e Redis).

Este trabalho implementou uma ferramenta própria de *benchmark* e a sua principal hipótese é que o tipo de dados influencia nas leituras e escritas, sendo utilizado portanto dois tipos de dados para as comparações, dados de sensores e dados de multimídia, ambos gerados por um gerador de dados sintético. Porém, este trabalho, somente realizou a comparação de desempenho em operações de escrita e leitura, utilizando os diferentes tipos de dados. Além disso, apesar de haver uma variação do tipo de dados, os dados foram organizados em uma simples linha, não sendo considerada diferentes hierarquias, como no presente trabalho.

3.2.2 NOSQL X NOSQL

O trabalho de Abramova et al. (2014) realizou a comparação de dez SGBD NoSQL, Cassandra, HBase, Oracle NoSQL, Redis, Scalaris, Tarantool, Voldemort, Elasticsearch, MongoDB e OrientDB, avaliando o tempo de carregamento dos dados e a velocidade de execução de diferentes tipos de requisição, utilizando para isso o YCSB em um ambiente não distribuído. Os testes foram realizados apenas sobre operações de leitura e escrita nas diferentes cargas de trabalho padrão do YCSB e em duas cargas estendidas. Porém, os dados utilizados pelo YCSB são organizados em uma estrutura simples, sem dados aninhados, formada por dez chaves e seus respectivos valores.

Já o trabalho de Gandini et al. (2014) realizou o benchmark entre os SGBD Cassandra, MongoDB e HBase, implantados sobre a plataforma em nuvem Amazon EC2, utilizando

diferente tipos de máquinas virtuais e tamanho dos clusters, visando estudar o efeito das diferentes configurações. Assim como o anterior, este trabalho utilizou o YCSB como framework para os testes, medindo tanto a latência, quanto o throughput, e os dados utilizados também possuíam um organização simples.

Diferentemente dos demais, o *benchmark* proposto por Abubakar et al. (2014) utilizou o YCSB em um ambiente onde os recursos eram limitados, sem utilizar *clusters* ou dados distribuídos, usando somente um PC e assumindo um *cluster* com somente um nó ou um sistema distribuído em um único desktop.

A comparação de desempenho neste trabalho foi realizada entre o MongoDB, ElasticSearch, Redis e OrientDB. Nas operações de inserção e atualização o Redis obteve melhores resultados. Já nas operações de leitura, para cada quantidade de dados utilizados nos testes, houve um SGBD que obteve melhores resultados, mas o ElasticSearch foi o que obteve um melhor desempenho na média destas operações.

Um Estudo do Impacto da Modelagem de Dados no Desempenho de Consultas nos SGBD NoSQL Orientados a Documentos

3.3 COMPARAÇÃO DOS TRABALHOS RELACIONADOS

A partir da descrição dos trabalhos relacionados é possível observar que a maioria das pesquisas relacionadas às modelagens de dados nos SGBD NoSQL não levaram em consideração a possibilidade de variar a modelagem de agregados de forma a otimizar as consultas analíticas. Além disso, em somente um deles foram apresentados resultados de desempenho a partir das modelagens propostas.

Entre os trabalhos relacionados que realizaram a comparação de desempenho entre os SGBD NoSQL, somente um deles mensurou a capacidade desses SGBD em realizar consultas analíticas. Porém, os testes também foram realizados sobre dados baseados em apenas uma única modelagem.

A partir das tabelas 3.1 e 3.2, que apresentam um resumo dos trabalhos relacionados, podemos observar que este trabalho se diferencia dos demais, em virtude de realizar um estudo do impacto da modelagem de agregados com a apresentação de resultados, utilizando diferentes modelagens de agregados, em 5 SGBD NoSQL distintos.

TAB. 3.1: Resumo de trabalhos relacionados envolvendo modelagem de dados

Trabalho	Foco	Testes	Variação da modelagem
(HOBERMAN, 2014)	Modelagem NoSQL para Web Services partindo de um modelo conceitual Referenciar as entidades ou aninhar os dados	Não	Não
(FOWLER; SADALAGE, 2013)	Modelagem NoSQL para Web Services - Referenciar as entidades ou aninhar os dados	Não	Não
(KAUR; RANI, 2013)	Modelagem NoSQL para Web Services partindo do modelo conceitual para Documentos e Grafos	Não	Não
(CHEVALIER et al., 2015)	Regras de mapeamento de esquemas conceituais multidimensionais para NoSQL	Tempo para contrução de treliças	Não
(SCABORA et al., 2016)	Impacto de diferentes configurações de famílias de colunas em SGBD orientados a colunas	Desempenho de consultas analíticas	Sim
(ELMASRI et al., 2005)	Modelagem XML – Teoria sobre utilizar diferentes dados na raiz dos documentos	Não	Não
(SCHROEDER; MELLO, 2008)	Modelagem XML – Melhor organização da estrutura hierárquica e uso de referências	Não	Não
(NAVATHE, 2014a)	Modelagem Orientada a Objeto - Mapeamento a partir do conceitual	Não	Não
(FONG, 1995)	Modelagem Orientada a Objeto - Mapeamento a partir do conceitual	Não	Não

TAB. 3.2: Resumo de trabalhos relacionados envolvendo benchmarks

Trabalho	Foco	Sist. Distribuído	Medições extras
(CARNIEL et al., 2012a)	Visões e Índices	Não	Espaço de Armazenamento das visões e indices
(PHAN et al., 2014)	Tipo de Dados	Não	Espaço de Armazenamento
(ABRAMOVA et al., 2014)	Workload	Sim	Tempo de Carregamento
(GANDINI et al., 2014)	Nº de núcleos/nós/ replicação	Sim	Throghput
(ABUBAKAR et al., 2014)	Ambiente Escasso	Não	Tempo de Carregamento
Um Estudo do Impacto da modelagem de dados no desempenho de consultas nos SGBD NoSQL Orientados a Documentos	**Diferentes Modelagens**	**Não**	**Espaço de Armazenamento**

4 IMPACTO DA MODELAGEM DE AGREGADOS E CRIAÇÃO DE HEURÍSTICAS

Neste capítulo será apresententado o estudo do comportamento dos diferentes SGBD NoSQL, com relação ao seu desempenho em consultas analíticas, conforme estas são realizadas em coleções com diferentes modelagens de agregados. Após o entendimento do comportamento de cada SGBD, será proposta uma heurística de escolha de modelagens de agregados, baseada em cada padrão de comportamento, de forma a aumentar a performance de consultas analíticas nos diferentes SGBD estudados.

4.1 JUSTIFICATIVA DO ESTUDO

Apesar de apresentarem aspectos importantes relacionados aos SGBD NoSQL, como visto no capítulo anterior, em nenhum trabalho relacionado foi explorada a capacidade de variar a modelagem dos agregados nos SGBD orientados a documentos, muito menos foram apresentadas análises a respeito do quanto a variação desta modelagem pode impactar no desempenho de consultas analíticas, havendo portanto uma lacuna na literatura sobre este tipo de SGBD que será preenchida em parte por este trabalho.

A partir da análise do impacto da variação da modelagem nos diferentes SGBD, em havendo aspectos que pesem no desempenho das consultas nesses SGBD, heurísticas de modelagem de agregados serão propostas de forma a auxiliar os modeladores de dados ou os desenvolvedores de sistemas de bancos de dados de forma a obterem um melhor desempenho dos SGBD orientados a documentos em consultas analíticas.

Na figura 4.1 podemos visualizar como um SGBD orientado a documentos poderá se beneficiar das heurísticas que são propostas mais adiante neste trabalho.

Acho que o H está embutido no construtor de agregados. É ele que tem a inteligência para, com base nas seletividades e outras estatísticas, selecionar o conjunto de agregados que é mais pertinente.

Segundo Garcia-Molina et al. (2000) o processador de consultas é a parte do SGBD que pode interferir na melhoria do desempenho das consultas. O processador de consultas possui dois componentes principais: O *compilador de consultas* e o mecanismo de execução. Na figura 4.1, o mecanismo de execução de consultas não aparece, pois as heurísticas propostas terão efeito principalmente no trabalho do *compilador de consultas*, que

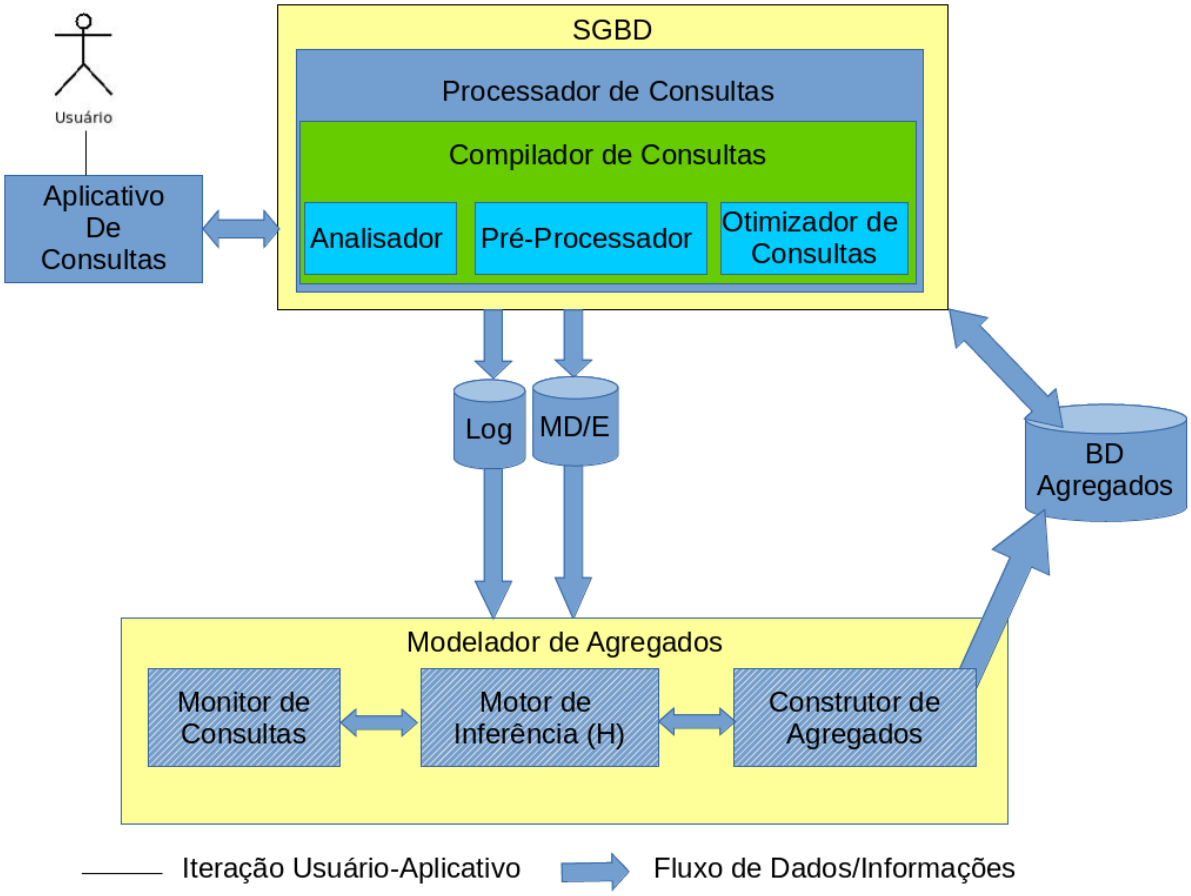

FIG. 4.1: Exemplo de utilização das heurísticas.

possui três unidades principais: *Analisador de consultas*, *Pré-processador de consultas* e *Otimizador de Consultas*.

Tipicamente, um *aplicativo de consultas* submete um conjunto de consultas analíticas a um SGBD orientado a documentos. O *analisador de consultas* gera *metadados* (*MD*) e *dados estatísticos* (*E*), sobre as consultas submetidas. A partir destes dados *o pré-processador* e o *otimizador de consultas* podem extrair uma série de informações, como tamanho estimado das coleções, se há índices úteis, etc, e com base nelas construir um plano de consultas mais eficiente para execução.

Este trabalho sugere incluir um módulo chamado *Modelador de Agregados*, ao processador de consultas de um SGBD. Esta funcionalidade a mais pode contribuir para a melhora do desempenho do mesmo: *Monitor de Consultas*, *Motor de Inferência* e *Construtor de Agregados*. A ideia é que os bancos de *Log* e de *metadados (MD)* podem fornecer ao *monitor de consultas* informações para que este identifique quais as consultas que mais precisam ter seu desempenho melhorado (consultas críticas). Com base nestas informações e se valendo das funções que executam as heurísticas que serão propostas neste trabalho,

o *Motor de Inferência* depreenderá quais modelagens de agregados mais favorecerão as consultas e irá fornecer instruções para que o módulo *Construtor de Agregados* construa os agregados identificados nos *bancos de agregados*.

A construção e a manutenção dos agregados em diferentes modelagens pode ocorrer dinamicamente à medida que os dados são alterados. O papel do *Motor de Inferência* consiste em estabelecer o momento em que as coleções de agregados devem ser construídas ou removidas dos *bancos de agregados*, seja pela mudança das consultas críticas, seja pela alteração dos dados.

Nos SGBD NoSQL, mais especificamente os orientados a documento a consistência é eventual, ou seja, a consistência dos dados será obtida de tempos em tempos. Portanto, o cenário acima descrito leva em consideração que as aplicações que utilizarão o sistema, não necessitem de informações atualizadas em um curto espaço de tempo, de forma que as atualizações dos dados dentro de um curto espaço de tempo não afetem as decisões a serem tomadas.

De forma mais simples, as heurísticas propostas poderão ser utilizadas por ocasião da migração dos dados de, por exemplo, um SGBD relacional, para um SGBD orientado a documentos. Ou ainda, servem para rever a modelagem de agregados inicial.

Com base em um conjunto pré-definido de consultas, é possível sugerir um conjunto inicial de modelagens para gerar as coleções de documentos. Uma vez que sejam disponibilizadas mais de uma coleção de documentos (diferentes modelagens), então o SGBD deve ser capaz de utilizar as heurísticas de redirecionamento de consultas para melhorar o desempenho das mesmas.

Assim, além de um amplo estudo sobre o comportamento dos SGBDs NoSQL orientados a documento, outra principal contribuição deste trabalho é a definição de um conjunto de heurísticas (H). Para obter e confirmar essas heurísticas, foi então necessário realizar um estudo, utilizando diferentes *benchmarks* (SSB e TPC-DS). Em linhas gerais, o que foi feito foi a geração de diferentes coleções de agregados (uma para cada modelagem possível), a partir dos dados de cada *benchmark*. Além disso, o conjunto de consultas críticas a nortear a geração dos agregados, foi baseado no conjunto pré-definido para cada *benchmark*. As seções a seguir explicam como foram montados os cenários dos experimentos realizados, e como foram interpretados seus resultados.

4.2 CRIAÇÃO DOS DADOS DO SSB E DOS AGREGADOS EM DIFERENTES MODELAGENS

Os SGBD NoSQL, por possuírem uma maior facilidade para lidar com dados distribuídos, permitem que à medida em que a quantidade de dados aumente, a capacidade computacional também aumente, de forma a manter o alto desempenho destes sistemas. Portanto, este trabalho realizará um estudo para uma quantidade de dados não muito grande, com o intuito de simular um único nó de um sistema, onde os dados possam caber na memória RAM de forma a permitir um melhor desempenho dos SGBD nas consultas, como recomendado pela documentação de alguns dos SGBD estudados[18].

Portanto, para os experimentos iniciais foi utilizado o gerador de conjunto de dados fornecido pelo SSB, a partir do qual foram gerados 1 GB de dados em um SGBD relacional.

A partir desses dados, em virtude de o SSB possuir quatro tabelas de dimensão e uma tabela de fatos, foi possível obter 5 diferentes modelagens de agregado de forma análoga à mostrada na subseção 2.4.

Nas figuras 4.2 a 4.6 podemos visualizar uma versão reduzida de um conjunto de dados do SSB, em agregados construídos com base em diferentes tabelas.

Repare que cada modelagem possui na raiz dos agregados os dados da tabela que serviu de base para a construção de cada agregado. A modelagem de agregado baseada na tabela *Part* da figura 4.3, por exemplo, possui na sua raiz os dados da tabela *Part* e os dados da tabela *Lineorder* estão aninhados a estes, pois é a única tabela que possui relacionamento direto com a tabela *Part*.

De forma análoga, os dados das outras tabelas de dimensão, que possuem relacionamentos com a tabela *Lineorder*, estão aninhados aos dados da tabela *Lineorder*.

Da mesma forma que a modelagem de agregado baseada na tabela *Part*, todas as outras modelagens de agregados baseadas nas tabelas de dimensão possuem três níveis de profundidade na hierarquia dos documentos. Já a modelagem de agregado baseada na tabela de fatos possui somente dois níveis, pois a tabela *Lineorder* possui relacionamento direto com todas as tabelas dimensão.

A partir desta seção iremos nos referir a cada modelagem e coleção de agregado conforme a tabela que foi utilizada como base para contrução do agregado. Por exemplo, ao nos referirmos à modelagem *Customer*, estaremos nos referindo à modelagem de agregado na qual os dados das raízes dos documentos são originados da tabela *Customer*, bem como, chamaremos cada coleção de documentos de acordo com a modelagem de agregado

[18]https://docs.mongodb.com/manual/faq/diagnostics/

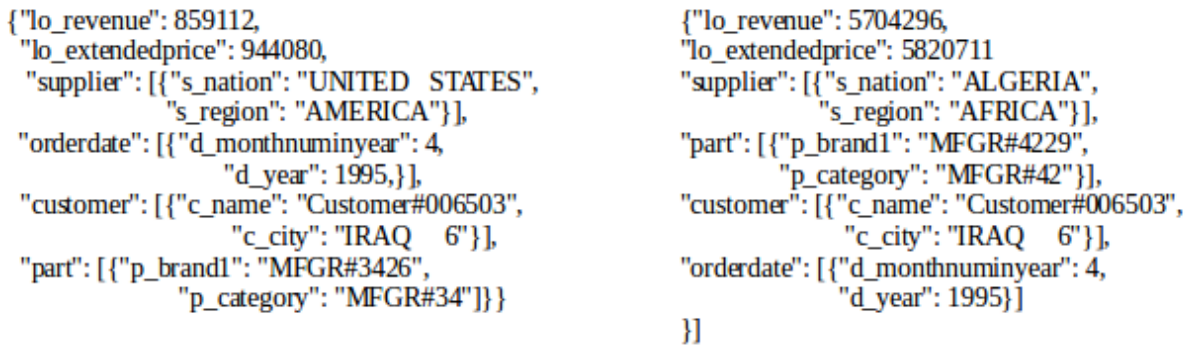

FIG. 4.2: Agregado com modelagem baseada na tabela *Lineorder*.

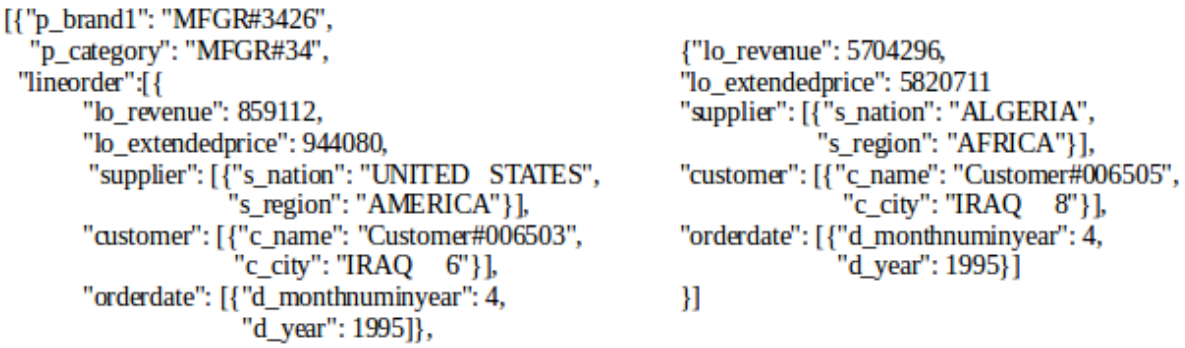

FIG. 4.3: Agregado com modelagem baseada na tabela *Part*.

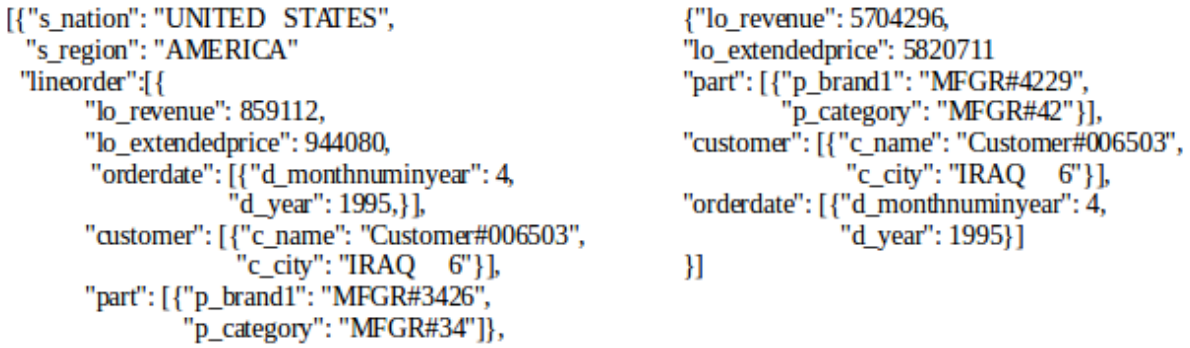

FIG. 4.4: Agregado com modelagem baseada na tabela *Supplier*.

dos documentos utilizada naquela coleção. A coleção *Customer*, por exemplo, é a coleção cujos agregados estão modelados conforme a modelagem *Customer*.

De forma análoga, quando utilizarmos o termo coleções baseadas nas tabelas de dimensão, estaremos nos referindo às coleções cujos dados nas raízes dos documentos são oriundos das tabelas de dimensão, assim como utilizaremos o termo coleção baseada na tabela de fato para nos referirmos à coleção cujos dados nas raízes dos documentos são oriundos da tabela de fato.

Embora o fator de escala utilizado neste trabalho tenha sido igual a 1, o que significa

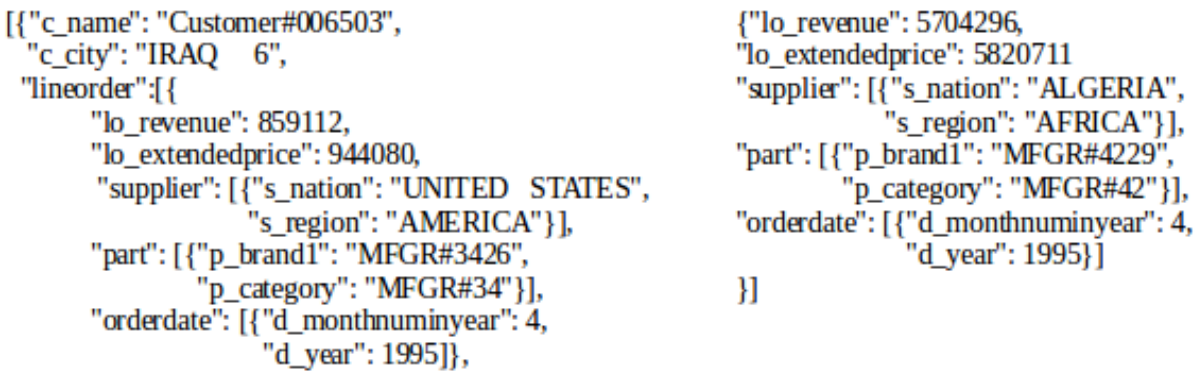

FIG. 4.5: Agregado com modelagem baseada na tabela *Customer*.

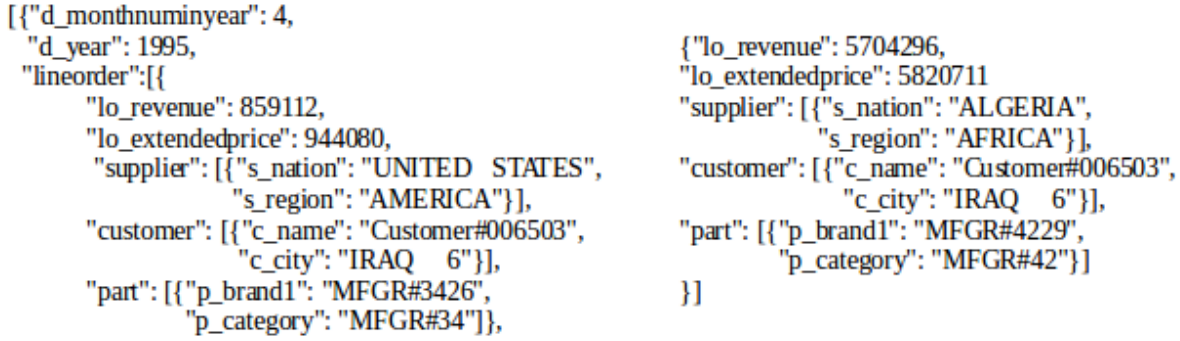

FIG. 4.6: Agregado com modelagem baseada na tabela Orderdate.

que foram gerados apenas 1 GB de dados, após a transformação dos dados existentes no banco de dados relacional para o formato JSON, as coleções de documentos criadas apresentaram um tamanho superior aos das tabelas de origem, principalmente devido à redundância de dados. Note que, na figura 4.6, diferentemente do que acontece em um banco relacional, os dados do cliente "Customer#006503" foram replicados nos dois pedidos feitos no mês 4, do ano de 1995.

O espaço de armazenamento utilizado para cada SGBD também foi diferente para cada coleção nas diferentes modelagens. A tabela 4.1 apresenta o tamanho dos documentos nas cinco coleções com diferentes modelagens, no formato JSON e o espaço ocupado pelos documentos em cada banco de dados. No caso do SGBD OrientDB, em virtude deste estudo utilizar tanto documentos aninhados como documentos ligados (como explicado na seção 2.2.3), e por estas formas de organizar os documentos apresentarem tamanhos de armazenamento distintos, o espaço ocupado por ambas as formas são apresentados.

O tamanho das coleções do ElasticSearch e do MarkLogic levaram em consideração o espaço ocupado também pelos índices criados automaticamente por estes SGBD e o espaço ocupado pelo CouchDB levou em consideração o espaço ocupado pelas visões

TAB. 4.1: Tamanho das coleções de agregados em Gigabytes no formato JSON e dentro de cada SGBD

Formato - SGBD	Customer	Orderdate	Part	Supplier	Lineorder
JSON	9,2	8,1	9,3	9,4	10,5
MongoDB	2,1	2	3	2,5	3,8
ElasticSearch	5,4	4,9	6,4	6,3	6,8
OrientDB documento	10	8,9	12,9	10,1	15,3
OrientDB ligados	13,3	12,1	14,9	13,8	16,9
CouchDB	2,3	2,1	2,8	2,7	7,5
MarkLogic	6,4	5,5	8,4	5,9	20,3

materializadas que foram geradas para resolver as consultas. O espaço ocupado pelas visões no CouchDB, neste trabalho, é pequeno se comparado ao espaço total ocupado pelos documentos. O espaço ocupado pelas treze visões materializadas no banco de dados da coleção *Customer*, por exemplo, foi de apenas 9 MB. Nos demais SGBD, o espaço apresentado corresponde apenas aos documentos.

4.3 AJUSTE DE CONSULTAS PARA CADA SGBD

O SSB possui no total treze consultas pré-definidas (Apêndice 1), sendo que estas consultas são divididas em quatro grupos, cuja complexidade vai aumentando a cada grupo de consulta.

Com a finalidade de demonstrar como uma consulta em SQL é expressa em cada linguagem de consulta dos diferentes SGBD em coleções com diferentes modelagens de agregado, iremos apresentar uma consulta do SSB na linguagem SQL e como esta pode ser expressa na linguagem dos diferentes SGBD, nas coleções *Part* e *Lineorder*.

A maioria das consultas padrão do SSB, realizam operações de ordenação, entretanto, como explicado na seção 2.2.2, o ES possui algumas limitações neste tipo de operação. Portanto, este tipo de operação foi retirado das consultas, de forma que as consultas em todos os SGBD fossem equivalentes.

Podemos visualizar na figura 4.7 a consulta 4.3 do SSB expressa na linguagem SQL e nas figuras seguintes a mesma consulta expressa nas linguagens dos diferentes SGBD estudados.

As figuras 4.8 e 4.9 representam essa mesma consulta expressadas na linguagem Aggregate-Framework do MongoDB, realizadas sobre os agregados baseados nas tabelas *Part* e *Lineorder*.

A linguagem de consulta do MongoDB utiliza uma sintaxe onde as operações são

```
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 p_category = "MFGR#14"
group by d_year, s_city, p_brand1;
```

FIG. 4.7: Consulta 4.3 do SSB na linguagem SQL.

```
db.part.aggregate([
{"$match":{"$and":[{"p_category":"MFGR#14"},{"lineorder.customer.c_region":"AMERICA"},
{"lineorder.supplier.s_nation": "UNITED STATES"},{"$or":[{"lineorder.orderdate.d_year":1997},
{"lineorder.orderdate.d_year":1998}]}]}},
{"$unwind":"$lineorder"},
{"$match":{"$and":[{"p_category":"MFGR#14"},{"lineorder.customer.c_region":"AMERICA"},
{"lineorder.supplier.s_nation": "UNITED STATES"},{"$or":[{"lineorder.orderdate.d_year":1997},
{"lineorder.orderdate.d_year":1998}]}]}},
{"$project":{"lineorder.orderdate.d_year":1,"lineorder.supplier.s_city":1,"p_brand1":1,
"dim":{"$subtract":["$lineorder.lo_revenue","$lineorder.lo_supplycost"]}}},
{"$group":{"_id":
{"d_year":"$lineorder.orderdate.d_year","s_city":"$lineorder.supplier.s_city","p_brand1":"$p_brand1"},
"profit":{"$sum": "$dim"}}}
]).toArray()
```

FIG. 4.8: Consulta 4.3 expressa na linguagem Aggregate Framework do MongoDB sobre a coleção *Part*.

```
db.lineorder.aggregate([
{$match:{$and:[{"customer.c_region":"AMERICA"},{"supplier.s_nation": "UNITED STATES"},{$or:
[{"orderdate.d_year":1997},{"orderdate.d_year":1998}]},{"part.p_category":"MFGR#14"}]}},
{$project:{"orderdate.d_year":1,"supplier.s_city":1,"part.p_brand1":1,
dim:{$subtract:["$lo_revenue","$lo_supplycost"]}}},
{$group:{_id:{d_year:"$orderdate.d_year",s_city:"$supplier.s_city",p_brand1:"$part.p_brand1"},
"profit":{$sum: "$dim"}}}
]).toArray()
```

FIG. 4.9: Consulta 4.3 expressa na linguagem Aggregate Framework do MongoDB sobre a coleção *Lineorder*.

realizadas uma após a outra como um *pipeline*. Repare na figura 4.8 que inicialmente é realizada a operação *match* que é equivalente à cláusula de seleção *WHERE* do SQL, havendo operadores booleanos *or* e *and* e operadores de comparação *lte* (<=) e *gte* (>=), dentro da cláusula de seleção. Posteriormente, é realizada a operação de *unwind* que é a operação de desagregação no MongoDB, seguida de uma nova operação *match*.

Ao final da consulta, é executada a operação *project* que equivale à cláusula de projeção *SELECT* em SQL, seguida de uma operação agrupamento de valores *group*, equivalente à operação *GROUP BY* em SQL. A última operação *toArray* tem a finalidade de apresentar todos os resultados em forma de um documento JSON, pois se esta operação não for utilizada, somente 20 resultados são apresentados.

Uma diferença que já se pode notar na expressão das consultas sobre coleções diferentes (baseadas em modelagens de agregados diferentes), é a necessidade ou não de se realizar a operação de desagregação conhecida no MongoDB como *unwind*. Quando a consulta é realizada sobre a coleção baseada na tabela de fatos, essa operação não é necessária. Isso pode fazer com que algumas consultas sejam favorecidas quando realizadas sobre essa coleção.

As figuras 4.10 e 4.11 mostram a expressão da linguagem Query-DSL do Elasticsearch para realizar a consulta 4.3 nas coleções baseadas nas tabelas *Part* e *Lineorder*. No ElasticSearch não é necessária a realização de uma operação de desagregação como no MongoDB, pois o ES trata os dados aninhados como documentos separados, que são ligados aos dados da raiz do documento por meio de mapeamento. Porém, para fazer uma consulta nos dados aninhados é necessário especificar o que a consulta está procurando nesses dados, operação esta conhecida como *inner hits* (GORMLEY; TONG, 2015), que pode ser vista na figura 4.10.

Assim como no MongoDB, na linguagem Query-DSL a operação *match* corresponde à cláusula de seleção, onde os operadores de booleanos *AND* e *OR* são expressos como *must* e *should* e os operadores de comparação são equivalentes aos do MongoDB, porém estes devem ser declarados como *range*.

O agrupamento no ElasticSearch é executado sequencialmente para cada valor agrupado. Na consulta 4.3, por exemplo, o agrupamento é feito sobre os campos "p_brand1", "d_year" e "s_city", porém o resultado mostrará os valores como um agregado, ou seja, dentro de cada valor de "p_brand1" serão apresentados todos os valores de "d_year" e dentro deste, todos os valores de "s_city", junto com a soma do campo "lo_revenue".

No ES não é possível realizar o agrupamento ou a ordenação de valores de um campo aninhado e um campo raiz, nesta ordem. Se um banco de dados possui agregados construídos utilizando a tabela *Part* como base, não será possível agregar por "d_year" e "p_category", mas somente por "p_category" e "d_year". Esta limitação foi o motivo da retirada da cláusula de ordenação das consultas.

Na figura 4.10 é demonstrada a expressão da consulta 4.3 do SSB na linguagem Query-DSL, para consultar os dados na modelagem *Part*. Note que, apesar de não ser necessário

```
curl -XPOST 'localhost:9200/i_part/part2/_search?pretty' -d '
{ "query": {"filtered": {"filter": {
  "bool": {"must": [{"match": {"p_category":"MFGR#14"}},{
   "filtered": {"filter": {
         "nested": {
          "inner_hits": {}, "path": "lineorder",
           "query": {"filtered": {"filter": {
             "bool": {"must": [
                       {"match": {"lineorder.customer.c_region":"AMERICA"}},
                       {"match": {"lineorder.supplier.s_nation":"UNITED STATES"}},
                       {"range": {"lineorder.orderdate.d_year":{"gte":1997,"lte":1998}}}
                            ]}}}}}}]}}}},
               "aggs": {"p_brand1": {"terms": {
                   "field": "p_brand1",
                     "order" : { "_term" : "asc" }, "size": 10000},
               "aggs": {"lineorder": {
                 "nested": {"path": "lineorder"},
               "aggs": {"only_loc": {
                 "filter": {
                   "bool": {"must": [
                       {"match": {"lineorder.customer.c_region":"AMERICA"}},
                       {"match": {"lineorder.supplier.s_nation":"UNITED STATES"}},
                       {"range": {"lineorder.orderdate.d_year":{"gte":1997,"lte":1998}}}]}},
               "aggs": { "year": {"terms": {
                "field": "lineorder.orderdate.d_year",
                  "order" : { "_term" : "asc" }, "size": 10000},
                "aggs": {"s_city": {"terms": {
                 "field": "lineorder.supplier.s_city",
                   "order" : { "_term" : "asc" }, "size": 10000},
                 "aggs": {"profit":{
                  "sum": {"script" : "doc[\"lineorder.lo_revenue\"].value -
                                      doc[\"lineorder.lo_supplycost\"].value"
                      }}}}}}}}}}}}
                      },"size":0
```

FIG. 4.10: Consulta 4.3 expressa na linguagem Query-DSL do Elasticsearch sobre a coleção *Part*.

realizar uma operação de desagregação no ES, é necessário consultar os dados aninhados duas vezes para que o resultado retorne os valores corretos, porém na modelagem baseada na tabela de fatos isto não é necessário, como pode ser visto na figura 4.11. Na figura 4.10 a consulta aos dados aninhados no ES, foi necessário utilizar o operador *inner_hits*, demostrando que os documentos aninhados devem ser consultados.

As figuras 4.12 e 4.13 mostram como é realizada a consulta 4.3 nas coleções *Part* e *Lineorder*, no dialeto GREMLIN do OrientDB. Da mesma forma que no MongoDB, no OrientDB é necessário realizar a operação de desagregação, também chamada de *unwind* neste SGBD, e da mesma forma que no MongoDB, na coleção que utiliza a modelagem de agregados baseada na tabela de fatos, esta operação não é necessária.

```
curl -XPOST 'localhost:9200/i_lineorder/lineorder/_search?pretty' -d '
{
    "aggs": {"only_loc": {
      "filter": {"bool": {
          "must": [
                {"term": {"supplier.s_nation":"UNITED STATES"}},
                {"term": {"part.p_category":"MFGR#14"}},
                {"term": {"customer.c_region":"AMERICA"}},
                {"range": {"orderdate.d_year":{"gte":1997,"lte":1998}}}
                  ]}},
      "aggs": {"year": {"terms": {
        "field": "orderdate.d_year",
         "order" : { "_term" : "asc" }, "size": 10000},
      "aggs": {"s_city": {"terms": {
        "field": "supplier.s_city",
         "order" : { "_term" : "asc" }, "size": 10000},
       "aggs": {"p_brand1": {"terms": {
        "field": "part.p_brand1",
         "order" : { "_term" : "asc" }, "size": 10000},
       "aggs": {"profit": {
          "sum": {"script" : "doc[\"lo_revenue\"].value - doc[\"lo_supplycost\"].value"}
                  }}}}}}}}}},"size":0
```

FIG. 4.11: Consulta 4.3 expressa na linguagem Query-DSL do Elasticsearch sobre a coleção *Lineorder*.

```
SELECT first(line.orderdate.d_year) as d_year, first(line.supplier.s_city) as s_city, p_brand1,
SUM(eval('line.lo_revenue.asLong() - line.lo_supplycost.asLong()')) as revenue
FROM (SELECT p_brand1, lineorder as line
      FROM part
      WHERE p_category="MFGR#14"
      AND lineorder.supplier contains(s_nation="UNITED STATES")
      AND lineorder.customer contains(c_region="AMERICA")
      AND lineorder.orderdate contains (d_year=1997 or d_year=1998) UNWIND line)
WHERE line.supplier contains(s_nation="UNITED STATES")
AND line.customer contains (c_region="AMERICA")
AND line.orderdate contains (d_year=1997 or d_year=1998)
GROUP BY line.orderdate.d_year, line.supplier.s_city, p_brand1
```

FIG. 4.12: Consulta 4.3 expressa na linguagem GREMLIN do OrientDB sobre a coleção *Part*.

No OrientDB, quando documentos ligados são utilizados, em consultas onde os atributos das raízes dos documentos não são utilizados, é possível realizar consultas diretamente sobre a classe *lineorder*, que representa os dados aninhados à classe que contém os dados das raízes dos documentos, sem a necessidade de realizar a operação *unwind*. A figura 4.14 é um exemplo de consulta à coleção *Customer* que evita a realização da operação *unwind*. Repare que não há na consulta nenhum atributo que pertence à raiz da modela-

```
SELECT first(part.p_brand1) as p_brand1, first(supplier.s_city) as s_city, first(orderdate.d_year) as
year, SUM(eval("lo_revenue.asLong() - lo_supplycost.asLong()")) as revenue
FROM lineorder
WHERE customer contains(c_region="AMERICA")
AND part contains(p_category="MFGR#14")
AND supplier contains(s_nation="UNITED STATES")
AND orderdate contains(d_year=1997 or d_year=1998)
GROUP BY orderdate.d_year, supplier.s_city, part.p_brand1
```

FIG. 4.13: Consulta 4.3 expressa na linguagem GREMLIN do OrientDB sobre a coleção *Lineorder*.

gem *Customer*.

Além disso, mesmo quando há atributos das raízes dos documentos nas cláusulas de seleção ou projeção da consulta, ainda é possível se beneficiar dessa facilidade, isto é, consulta direta sobre a classe aninhada. Isso é possível devido ao recurso de criação de relacionamentos reversos. Estes relacionamentos são novas ligações entre os elementos da classe aninhada e os elementos da classe raiz respectivos. A consulta da figura 4.15 é um exemplo de consulta que utiliza o relacionamento reverso denominado ”when” para obter os valores do campo ”d_year”, que é um campo da raiz da coleção *Orderdate*.

```
SELECT sum(lo_revenue.asLong()), first(orderdate.d_year), first(part.p_brand1)
FROM lineorder
WHERE supplier contains(s_region="AMERICA")
AND part contains(p_category ="MFGR#12")
GROUP BY orderdate.d_year, part.p_brand1
```

FIG. 4.14: Consulta 2.1 expressa na linguagem GREMLIN do OrientDB sobre a coleção *Customer* diretamente na classe *lineorder*.

```
SELECT sum(lo_revenue.asLong()), when.d_year, first(part.p_brand1)
FROM lineorder
WHERE part contains(p_category ="MFGR#12") AND
supplier contains(s_region="AMERICA")
GROUP BY when.d_year, part.p_brand1
```

FIG. 4.15: Consulta 2.1 expressa na linguagem GREMLIN do OrientDB sobre a coleção *Orderdate* utilizando relacionamentos reversos.

As figuras 4.16 e 4.17 representam a função Map-Reduce na linguagem Javascript utilizada pelo CouchDB para construir as visões materializadas para a consulta 4.3 do SSB nas coleções *Part* e *Lineorder* e as figuras 4.18 e 4.19 representam a forma na qual as visões são consultadas.

A função Map-Reduce percorre documento a documento, testando as cláusulas de seleção, partindo dos dados da raiz, se as condições dos dados da raiz são satisfeitas os dados aninhados são testados e assim em diante até alcançar o último nível da estrutura, não havendo portanto a necessidade de realizar uma operação de desagregação.

Após analisar todos os documentos, se as condições de seleção forem satisfeitas, os campos que farão parte da visão materializada e seus respectivos valores são selecionados.

```
{
"language": "javascript",
        "views": {"q43": {
                "map": "function(doc)  {var p_category = doc.p_category
                        var p_brand1 = doc.p_brand1
                        if (p_category == \"MFGR#14\"){
                                for each(lineorder in doc.lineorder) {
                                        for each(customer in lineorder.customer) {
                                        var c_region = customer.c_region}
                                        if (c_region == \"AMERICA\"){
                                                for each(supplier in lineorder.supplier) {
                                                var s_city = supplier.s_city
                                                var s_nation = supplier.s_nation}
                                                if (s_nation == \"UNITED STATES\"){
                                                        for each(orderdate in lineorder.orderdate) {
                                                                var d_year = orderdate.d_year}
                                                        if (d_year == 1997 || d_year == 1998) {
                                        emit({d_year: d_year,s_city: s_city, p_brand1: p_brand1},
                                        lineorder.lo_revenue - lineorder.lo_supplycost);
                                                                }}}}}}",
                        "reduce": "_sum"
                }}}
```

FIG. 4.16: Consulta 4.3 expressa na linguagem Javascript utilizando a função Map-Reduce sobre a coleção *Part*.

Como o objetivo deste trabalho é investigar o impacto das modelagens de agregado no desempenho das consultas, as visões materializadas foram geradas para atender especificamente as consultas do SSB.

As figuras 4.20 e 4.21 representam a consulta 4.3 nas coleções *Part* e *Lineorder* expressas na linguagem XQuery 1.0 que é uma das linguagens do MarkLogic e que foi utilizada neste trabalho, podendo ainda ser utilizadas outras linguagens mencionadas na seção 2.2.5.

O XQuery 1.0 pode ser utilizado tanto para documentos em JSON, quanto para documentos em XML. O XQuery 1.0 original não realiza a operação *group by*, porém no MarkLogic é possível realizar esta operação por meio da função *Map*. Esta função possibilita a realização de operações de agregação em chaves que possuem o mesmo valor. Na figura 4.20 podemos notar que para realizar a operação de agregação no foi necessário

```
{
"language": "javascript",
"views": {"q43": {"map": "function(doc)  {
                        for each(part in doc.part) {
                                var p_category = part.p_category
                                var p_brand1 = part.p_brand1
                                if (p_category == \"MFGR#14\"){
                                        for each(customer in doc.customer) {
                                                var c_region = customer.c_region}
                                        if (c_region == \"AMERICA\"){
                                                for each(supplier in doc.supplier) {
                                                        var s_city = supplier.s_city
                                                        var s_nation = supplier.s_nation}
                                                if (s_nation == \"UNITED STATES\"){
                                                        for each(orderdate in doc.orderdate) {
                                                                var d_year = orderdate.d_year}
                                                        if (d_year == 1997 || d_year == 1998) {
                                                        emit({  d_year: d_year,s_city: s_city, p_brand1:
                                                p_brand1}, doc.lo_revenue - doc.lo_supplycost);
                                                        }}}}}}",
                "reduce": "_sum"
                }}}
```

FIG. 4.17: Consulta 4.3 expressa na linguagem Javascript utilizando a função Map-Reduce sobre a coleção *Lineorder*.

```
curl -X GET http://127.0.0.1:5984/db_part/_design/query43/_view/q43?group=true
```

FIG. 4.18: Consulta 4.3 sobre a visão materializada gerada na coleção *Part* no CouchDB.

```
curl -X GET http://127.0.0.1:5984/db_lineorder/_design/query43/_view/q43?group=true
```

FIG. 4.19: Consulta 4.3 sobre a visão materializada gerada na coleção *Lineorder* no CouchDB.

concatenar os valores dos campos da cláusula de projeção (operação concat()) em uma chave ($key) e para cada valor igual de chaves é realizado o agrupamento.

As demais consultas do SSB podem ser encontradas no Apêndice 1 e as adaptações de todas as consultas utilizadas neste trabalho, para todas as linguagens dos diferentes SGBD, não serão expostas neste trabalho, pois o número de páginas necessário para isto seria por volta de 180 páginas, considerando as consultas utilizadas por ocasião da validação dos resultados e das consultas utilizando índices. Porém, as consultas do SSB e as adaptações para cada SGBD em cada modelagem de agregado podem ser consultadas no endereço eletrônico: https://github.com/raphha/Doc_Store_SSB.

```
xquery=
xquery version "1.0-ml";
<query43>{
 let $m := map:map()
 let $build :=
   for $doc in doc()[p_category="MFGR#14"]
   for $sales in $doc/lineorder[supplier/s_nation="UNITED STATES" and
customer/c_region="AMERICA" and (orderdate/d_year=1997 or orderdate/d_year=1998)]
   let $s_city:=$sales/supplier/s_city
   let $p_brand1:=$doc/p_brand1
   let $d_year:= $sales/orderdate/d_year
   let $key := concat($d_year, ',', $s_city, ',', $p_brand1)
   return map:put(
    $m, $key, sum((
      map:get($m, $key),(
      xs:long($sales/lo_revenue)-xs:long($sales/lo_supplycost)))))
 for $key in map:keys($m)
 return
 <group>{concat($key,',',map:get($m, $key))}</group>
}</query43>
```

FIG. 4.20: Consulta 4.3 na linguagem de consulta do MarkLogic sobre a coleção *Part*.

```
xquery=
xquery version "1.0-ml";
<query43>{
 let $m := map:map()
 let $build :=
   for $sales in doc()[supplier/s_nation="UNITED STATES"
and customer/c_region="AMERICA" and (orderdate/d_year=1997 or orderdate/d_year=1998) and
part/p_category="MFGR#14"]
   let $s_city:=$sales/supplier/s_city
   let $p_brand1:=$sales/part/p_brand1
   let $d_year:= $sales/orderdate/d_year
   let $key := concat($d_year, ',', $s_city, ',', $p_brand1)
   return map:put(
    $m, $key, sum((
      map:get($m, $key),(
      xs:long($sales/lo_revenue)-xs:long($sales/lo_supplycost)))))
 for $key in map:keys($m)
 return
 <group>{concat($key,',',map:get($m, $key))}</group>
}</query43>
```

FIG. 4.21: Consulta 4.3 na linguagem de consulta do MarkLogic sobre a coleção *Lineorder*.

4.4 CONFIGURAÇÕES PARA OS EXPERIMENTOS

Para os experimentos foi utilizado um Notebook da ASUS com 16GB de memória RAM e um HD com 1TB e 64mb de Cache com 5400 Rpm, com 8 processadores 2.1GHz.

As versões utilizadas para os experimentos foram as seguintes versões gratuitas de cada SGBD:

- MongoDB: 3.2.4
- ElasticSearch: 2.2.1

- OrientDB: 2.1.15
- CouchDB: 1.6.1
- MarkLogic: 8

Tanto o MongoDB, o ElasticSearch, o CouchDB e o MarkLogic foram testados com as configurações padrões de instalação, a saber:

O MongoDB utiliza tanto o Wired Tiger cache quanto o filesystem cache. O Wired Tiger utiliza 60% da memória RAM menos 1GB, ou seja, na máquina testada utilizará 8,6 GB e o filesystem cache utiliza todo restante da memória RAM não utilizada pelo Wired Tiger.

Já o Elasticsearch utiliza 1GB como máximo da memória heap e 256MB como mínimo.

O CouchDB utiliza o sistema operacional para gerenciar os dados, portanto quanto maior a memória RAM melhor o desempenho, pois o CouchDB faz uso constante dos dados em cache (filesystem cache).

A primeira vez que o MarkLogic é executado, ele reserva para si o máximo de memória RAM disponível[19].

Já para o OrientDB foi necessário realizar alguns ajustes. Foi preciso aumentar a memória RAM disponível para este SGBD de 4GB para 8GB e prover uma memória heap mínima de 512MB e máxima de 6GB, de modo a viabilizar a execução de algumas consultas, pois não foi possível executá-las a partir das configurações padrão, devido ao erro recorrente de falta de memória heap ou pelo erro de GC Overhead Limit Exceed (GCOLE). Apesar da alteração das configurações padrão no OrientDB, o último erro citado ainda ocorreu em algumas consultas, onde a modelagem de agregado não favorecia a consulta. Nos resultados apresentados mais adiante o motivo da ocorrência deste erro será melhor explicado.

4.5 ENTENDIMENTO DOS RESULTADOS APRESENTADOS

Nas próximas seções deste capítulo serão discutidos os resultados dos experimentos iniciais e discutidos os comportamentos dos SGBD nas diferentes modelagens de agregado que nortearão a construção de heurísticas de escolha de modelagem de agregados. Portanto, é necessário o entendimento de como foram feitas as medições, e a maneira como os resultados serão apresentados.

[19] https://docs.marklogic.com/guide/installation/intro

No estudo do comportamento de cada SGBD, será apresentada uma tabela com a média de cinco execuções sequenciais de cada consulta, sobre as diferentes coleções, excluindo-se a primeira execução da consulta, onde os números em destaque (em amarelo) representam o tempo de execução da consulta na coleção com a modelagem que mais a favoreceu. Na tabela 4.2, por exemplo, a consulta 1.1 teve a sua melhor performance na coleção *Lineorder*.

Os tempos das primeiras execuções das consultas não serão utilizados para compreensão dos comportamentos dos SGBD, em virtude das primeiras consultas não apresentarem um padrão de comportamento na maioria dos SGBD. Entretanto, por ocasião da comparação de desempenho entre os SGBD, estes serão utilizados.

A maneira mais comum de modelar um agregado, conforme a encontrada em alguns trabalhos como o de Carniel et al. (2012b), é aquela onde os dados da tabela de fato estão nas raízes dos documentos, ou seja, para o conjunto de dados do SSB, a modelagem mais comum é a modelagem *Lineorder*. Portanto, com a finalidade de enfatizar os ganhos obtidos pelas coleções baseadas nas tabelas de dimensão, em relação à coleção baseada na tabela de fato, foi acrescentada uma coluna com o ganho obtido entre o tempo de execução da consulta na coleção que teve o melhor desempenho na consulta, e o tempo de execução da mesma consulta na coleção baseada na tabela de fato.

Na tabela 4.2, por exemplo, como o tempo de execução da consulta 1.2 foi menor na coleção *Orderdate*, e o ganho de 57% refere-se ao ganho de tempo obtido pela execução desta consulta na coleção *Orderdate* com relação ao tempo de execução na coleção *Lineorder*.

Para estudar melhor o comportamento dos SGBD com relação às consultas e coleções, tomou-se como base uma das heurísticas de desempenho usada tradicionalmente nos mecanismos de otimização dos SGBD (NAVATHE, 2014b), que usa a seletividade da condição da consulta, para escolher melhores planos para execução da mesma. A tabela 4.3 mostra a seletividade dos campos raízes (definida na Seção 2.1.5) nas consultas do SSB, ou seja, a SCR de cada consulta. Na consulta 2.1, por exemplo, a cláusula de seleção (cláusula WHERE) possui apenas os campos *p_category* e *s_region*, que são campos raízes das coleções *Part* e *Supplier*, respectivamente. Portanto, na linha da consulta 2.1 desta tabela, somente essas coleções terão as SCR apresentadas.

A partir da análise do comportamento dos diferentes SGBD, serão propostas heurísticas de modelagem de agregados, que poderão ser utilizadas em aplicações que realizam consultas analíticas, de modo a obter melhores desempenhos. As heurísticas propostas foram obtidas com base no *benchmark* SSB e confirmadas pelo *benchmark* TPC-DS. Assim,

devem ser válidas para bancos dados similares.

Para cada heurística criada será utilizada uma sigla para facilitar a referência às heurísticas.

4.6 RESULTADOS E HEURÍSTICAS NO MONGODB

A partir da tabela 4.2 é possível observar que cada consulta no MongoDB foi favorecida por uma coleção diferente, fazendo-se necessário realizar um estudo de quais aspectos, relacionados às consultas, foram predominantes para que certas coleções com diferentes modelagens de agregados tivessem melhor desempenho que outros em consultas específicas.

TAB. 4.2: Média do tempo de execução das consultas do SSB no MongoDB em milissegundos.

Consulta	Lineorder	Customer	Orderdate	Part	Supplier	Ganho
1.1	16307	97017	22324	89706	133354	0%
1.2	12238	66995	5259	38310	128515	57%
1.3	12655	80019	4620	49875	134238	63%
2.1	12733	97816	119676	9197	26783	28%
2.2	13763	101430	121632	5952	31265	57%
2.3	12130	34974	111300	5295	26533	56%
3.1	15232	26212	114005	104220	34865	0%
3.2	12197	8558	109198	59190	9354	30%
3.3	15979	4459	117799	21428	8282	72%
3.4	15980	4119	8197	17950	5543	74%
4.1	18515	23273	120152	39764	27480	0%
4.2	18818	23460	32394	40598	27255	0%
4.3	17262	22951	31828	12411	13021	28%
Média	14909	45483	70645	37992	46653	36%

Observando as tabelas 4.2 e 4.3, é possível notar que quando as coleções baseadas nas tabelas de dimensão apresentaram as SCR menores ou iguais a 0,04, as consultas sobre estas coleções foram mais rápidas que as consultas sobre a coleção *Lineorder*, e quando a SCR foi maior que 0,142 as consultas na coleção *Lineorder* foram mais rápidas. Portanto, existe um valor de corte para a SCR, que deve estar entre 0,04 e 0,142, abaixo do qual, as coleções baseadas nas tabelas de dimensão irão ter melhor desempenho nas consultas que a coleção *Lineorder*.

Com base nessa observação, buscou-se então encontrar um valor de corte único, que poderia ser usado na heurística. Para isso, realizou-se um pequeno estudo sobre o comportamento das coleções.

TAB. 4.3: Seletividade dos campos raízes das coleções do SSB, e no final, o número de documentos em cada coleção de agregado.

Consultas	Lineorder	Customer	Orderdate	Part	Supplier
1.1	0,166		0,142		
1.2	0,045		0,011		
1.3	0,045		0,002		
2.1				0,04	0,2
2.2				0,008	0,2
2.3				0,001	0,2
3.1		0,2	0,875		0,2
3.2		0,04	0,875		0,04
3.3		0,008	0,875		0,008
3.4		0,008	0,011		0,008
4.1		0,2		0,4	0,2
4.2		0,2	0,25	0,4	0,2
4.3		0,2	0,25	0,008	0,04
Documentos	6001215	30000	2556	200000	2000

Para este estudo foram realizadas consultas sobre as coleções baseadas nas tabelas de dimensão, utilizando nas consultas um filtro que possuía uma determinada SCR e a mesma consulta, utilizando o mesmo filtro foi realizada na coleção baseada na tabela de fatos. As duas consultas abaixo representam um exemplo de uma consulta nas coleções *Customer* e *Lineorder* cujo filtro na coleção *Customer* possui uma SCR igual a 0,016.

- **Consulta sobre a coleção *Customer*:**

```
'db.customer.aggregate([{"$match":{"c_custkey":
{"$gte":0,"$lte":500}}},
{"$unwind": "$lineorder"},
{"$match":{"c_custkey":{"$gte":0,"$lte":500}}},
{"$group":{"_id":null, "total":{"$sum":1}}}]).toArray()'
```

- **Consulta sobre a coleção *Lineorder*:**

```
'db.lineorder.aggregate([{"$match":{"customer.c_custkey":
{"$gte":0,"$lte":1}}},
{"$group":{"_id":null, "total":{"$sum":1}}}]).toArray()'
```

Os gráficos da figura 4.22 apresentam curvas de tempo de execução de consultas, sobre coleções baseadas nas tabelas de dimensão e sobre a coleção baseada na tabela de fato *Lineorder* em função da SCR de cada coleção baseada nas tabelas de dimensão.

A partir desses gráficos é possível observar que para valores de SCR mais baixos, o tempo de execução das consultas nas coleções de dimensão é menor que o da coleção baseada na tabela de fatos. No entanto, à medida em que a SCR aumenta, o tempo nas coleções de dimensão passa a ficar maior que o tempo na coleção baseada na tabela de fato, havendo um ponto onde as curvas se encontram. O ponto de encontro obtido por meio das curvas foi diferente para cada coleção, fazendo com que, para cada coleção, haja um valor de SCR, abaixo do qual esta coleção será mais rápida do que a coleção *Lineorder*. Apesar do valor de encontro das curvas ter ocorrido em valores de SCR diferentes para cada coleção, os pontos de encontro ocorreram entre valores próximos a 0,11 e 0,05, dentro do intervalo (0,04 a 0,142) observado anteriormente. Assim, com base nesses valores, foi possível arbitrar um valor de corte único (0,1) a ser usado para todas coleções baseadas na tabela de dimensão.

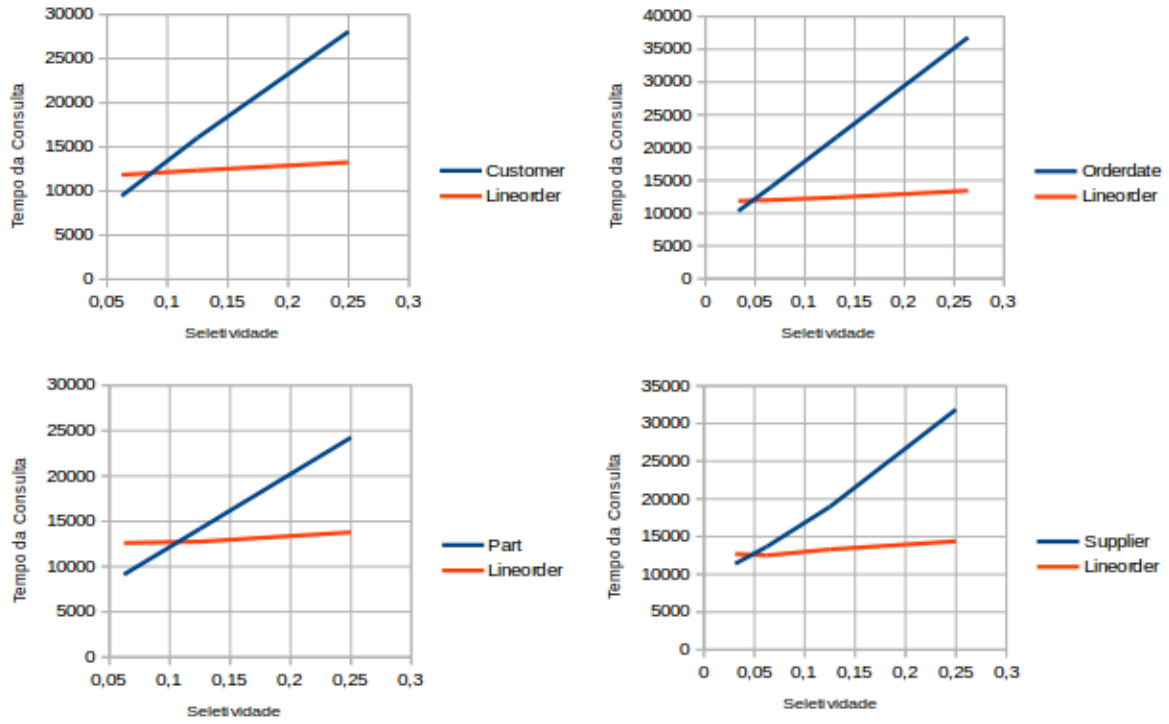

FIG. 4.22: Gráficos Tempo x Seletividade no MongoDB.

O motivo para que as coleções baseadas nas tabelas de dimensão tenham melhor desempenho quando a SCR é baixa, é devido ao fato de que o número de documentos nos quais é necessário verificar as condições de seleção dos campos que não pertencem à raiz é menor, pois se os filtros dos campos raízes não são satisfeitos, os outros filtros das consultas não são verificados. Além disso, o número de documentos que satisfazem às condições de seleção de cada consulta também é proporcional à SCR. Portanto, como

um número menor de documentos é selecionado, será necessário realizar menos operações *unwind*, fazendo com que estas coleções sejam mais eficientes que a coleção baseada na tabela de fatos.

Para o conjunto de dados do SSB, a partir da seletividade de corte de cada coleção baseada nas tabelas de dimensão foi possível utilizar um valor de corte único para o seu conjunto de consulta. Porém, estes valores de corte foram obtidos através de experimentos, para este conjunto/volume de dados específico. Por isso, este valor de corte não pode ser generalizado para todos os conjuntos de dados em diferentes volumes.

A consulta 2.3 na tabela 4.2, por exemplo, teve uma melhor performance na coleção *Part*, devido a esta consulta ter um filtro em um campo que está na raiz dos documentos da coleção *Part*, e pela seletividade deste campo ser de 0,001 aproximadamente. Esta baixa seletividade permitiu que a operação *unwind* tivesse que ser realizada somente nos documentos que satisfizessem aquele filtro.

As seletividades dos campos aninhados em níveis abaixo da raiz dos documentos também influenciam o número de documentos que devem executar a operação *unwind*, porém os resultados mostraram que a seletividade dos campos que estão nas raízes dos documentos foi o aspecto que mais influenciou no desempenho das consultas. Logo, aquelas seletividades não foram consideradas neste estudo.

Nas consultas 1.1, 3.1, 4.1 e 4.2 da tabela 4.2, a SCR em todas as coleções baseadas nas tabelas de dimensão eram maiores que 0,1, logo nesses casos, a coleção *Lineorder*, obteve a melhor performance. Este comportamento justifica-se pelo fato de que mesmo tendo que realizar a procura em todos os documentos da coleção, a operação *unwind* não é necessária nesta coleção.

Outra observação interessante nos resultados das consultas do SSB e nos gráficos foi que, quando duas coleções baseadas em tabelas de dimensão tinham SCR iguais, a coleção que continha um maior número de documentos obteve um tempo menor na consulta. Esse fato também contribuiu para que o ponto de encontro das curvas de Tempo *versus* Seletividade ocorresse em valores de seletividade mais baixos nas coleções *Supplier* e *Orderdate* que possuíam menor número de documentos.

Em coleções com maior número de documentos, a concentração de campos aninhados em um único documento é menor, pois os campos aninhados estão mais espalhados nos documentos. Portanto, o número de campos aninhados que estarão presentes na operação *unwind* será menor, o que faz com que a consulta seja mais rápida na coleção com maior número de documentos, para uma mesma SCR. A consulta 3.2 é um exemplo deste comportamento, onde ambas as coleções *Customer* e *Supplier* tinham uma seletividade

de 0,04, mas devido à coleção *Customer* ter um número maior de documentos, esta obteve um tempo menor nesta consulta.

A partir dos resultados das consultas do SSB no MongoDB, utilizando diferentes modelagens de agregados foi possível obter ganhos de até 74% em uma única consulta e uma média de ganho de 36%, ao se comparar os melhores tempos obtidos em cada consulta nas tabelas de dimensão com os tempos das consultas na coleção *Lineorder*.

Estes resultados mostram o quão importante é a escolha de uma modelagem de agregado para as consultas e sugere a manutenção de mais de uma coleção em diferentes modelagens, para o mesmo conjunto de dados, com o objetivo de prover uma melhor performance para um número maior de consultas.

Baseado nos experimentos no MongoDB e na análise do comportamento dos tempos obtidos nas consultas em diferentes modelagens de agregado, é possível propor uma heurística de escolha de modelagens de agregados neste SGBD.

Para apresentarmos a heurística do MongoDB é necessário definirmos alguns termos que serão utilizados na sua formalização.

Sejam:

- $C^D = \{C^{D_1}, ..., C^{D_n}\}$ o conjunto definido na seção 2.1.2, formado pelas coleções com modelagens baseadas nas tabelas de dimensões D_1 a D_n do esquema E;
- $\eta(x)$ o número de documentos existentes em uma coleção $x \in C^D$;
- $Q = \{Q_1, ..., Q_m\}$ um conjunto de consultas analíticas realizadas por um sistema ou aplicativo;
- $\sigma_{Q_i}(x)$ o valor da seletividade dos campos raízes (conforme a seção 2.1.5) de uma coleção $x \in C^D$ em uma consulta Q_i. Se uma coleção x não possui um campo raiz na cláusula de seleção da consulta Q_i, então $\sigma_{Q_i}(x) = 1$;
- C^F a coleção com modelagem de agregado baseada na tabela de fatos F, definida conforme a seção 2.1.2;
- R o conjunto de coleções escolhidas para incrementar o desempenho das consultas em Q, onde no início da heurística, $R = \{\}$;
- Q^A o conjunto de consultas de Q associadas[20] a uma coleção em R;

[20]Uma consulta associada a uma coleção é uma consulta que possui o melhor desempenho na coleção na qual foi associada

- L^x o valor de corte estimado para uma coleção $x \in C^D$;
- Q_L^x o conjunto de consultas pertencentes a Q na qual uma coleção $x \in C^D$ possui o valor de SCR menor que L^x;
- Q^x o conjunto de consultas associadas a $x \in R$ após a aplicação da heurística; e
- Ordena(R) é uma função que ordena o conjunto de coleções contidas em R, utilizando como primeiro critério o menor número de consultas na qual cada coleção de R possui a SCR menor que a de corte arbitrada. Havendo duas coleções empatadas neste primeiro critério, o segundo critério de ordenação é o menor número de consultas que cada coleção favorece.

Portanto, a heurística do MongoDB pode ser visualizada no algorítimo 1.

Para facilitar a explicação do algoritmo que representa a heurística proposta, este foi dividido em cinco passos.

No passo (i) da heurística, o conjunto de coleções R que devem otimizar as consultas em Q será inicializado como um conjunto vazio.

No passo (ii) é adicionado ao conjunto Q_L^x de cada coleção x pertencente a C^D, as consultas pertencentes a Q, se a SCR ($\sigma_{Q_k}(x)$) for menor que a seletividade de corte (L^x) daquela coleção.

Ainda neste passo, se a SCR de uma coleção x for menor que a SCR de todas as outras coleções y existentes em C^D ou o número de documentos desta coleção for maior do que das outras coleções que possuem a mesma SCR na referida consulta, a coleção x é adicionada ao conjunto R. Para cada consulta de Q^k em que ocorrer a situação acima, esta consulta é adicionada ao conjunto de consultas associadas à coleção x (Q^x) e ao conjunto Q^A, ou seja, é sinalizado que este tipo de consulta terá um melhor desempenho em uma coleção x existente em R.

Neste trabalho foi utilizado o princípio de Pareto (KOCH, 2000) como base para a seleção de um mínimo de modelagens que satisfizessem a maioria das consultas, de modo a evitar o custo de manutenção de todas coleções. Ou seja, manter coleções que permitem uma melhoria da performance de apenas 20% das consultas seria contraprodutivo.

Portanto, no início do passo (iii) o conjunto R é ordenado segundo a função Ordena(), definida anteriormente e após isso serão retiradas de R as coleções que favorecem menos de 20% das consultas ($|Q^x|/|Q| < 0,2$). As consultas que estavam associadas a cada coleção retirada de R, são retiradas do conjunto Q^A. No final do passo (iii) do algoritmo, é verificado se existe outra coleção baseada em uma tabela de dimensão com a SCR menor

Algorithm 1 Heurística do MongoDB (M1).

Entrada: C^D, $\eta(C^{D_i})$, Q, $\sigma_{Q_n}(C^{D_i})$, C^F, $L^{C^{D_i}}$
Saida: R; Q^x,$\forall x \in C^D$; Q^{C^F}
função seleciona_modelagens(C^D, C^F, $\eta(C^{D_i})$, Q, $\sigma_{Q_n}(C^{D_i})$, $L^{C^{D_i}}$)
Passo(i):
    $R \leftarrow \{\}$;
Passo(ii):
    Para cada $x \in C^D$, faça:
        Para cada $Q_k \in Q$, faça:
            Se $\sigma_{Q_k}(x) < L^x$ então:
                $Q^x_L \leftarrow Q^x_L \cup \{Q_k\}$;
                existe_melhor $\leftarrow 0$
                Para cada $y \in C^D$, onde $y \neq x$, faça:
                    Se ($\sigma_{Q_k}(y) < \sigma_{Q_k}(x)$) ou ($\sigma_{Q_k}(y) = \sigma_{Q_k}(x)$ e $\eta(y) > \eta(x)$) então:
                        existe_melhor $\leftarrow 1$
                        Pare;
                Se !existe_melhor então:
                    $R \leftarrow R \cup \{x\}$;
                    $Q^x \leftarrow Q^x \cup \{Q_k\}$;
                    $Q^A \leftarrow Q^A \cup \{Q_k\}$;
Passo(iii):
    $R \leftarrow$ Ordena(R);
    Para cada $x \in R$, faça:
        Se $|Q^x|/|Q| < 0,2$ então:
            $R \leftarrow R - \{x\}$
            $Q^A \leftarrow Q^A - Q^x$
            Para cada $z \in R$, faça:
                Para cada $Q_k \in Q^x$, faça:
                    Se $\sigma_{Q_k}(z) < L^z$ então:
                        existe_melhor $\leftarrow 0$;
                        Para cada $y \in R$, onde $y \neq z$, faça:
                            Se ($\sigma_{Q_k}(y) < \sigma_{Q_k}(z)$) ou ($\sigma_{Q_k}(y) = \sigma_{Q_k}(z)$ e $\eta(y) > \eta(z)$) então:
                                existe_melhor $\leftarrow 1$;
                                Pare;
                        Se !existe_melhor então:
                            $Q^z \leftarrow Q^z \cup \{Q_k\}$;
                            $Q^A \leftarrow Q^A \cup \{Q_k\}$;
Passo(iv):
    Se $|Q^A|/|Q| < 0,8$ então:
        $R \leftarrow R \cup \{C^F\}$;
Passo(v):
    Se $C^F \in R$ então:
        $Q^{C^F} \leftarrow Q - Q^A$;
    Senão:
        Para cada $x \in R$, faça:
            Para cada $Q_k \in Q - Q^A$, faça:
                existe_melhor $\leftarrow 0$
                Para cada $y \in R$, onde $y \neq x$, faça:
                    Se ($\sigma_{Q_k}(y) < \sigma_{Q_k}(x)$) ou ($\sigma_{Q_k}(y) = \sigma_{Q_k}(x)$ e $\eta(y) > \eta(x)$) então:
                        existe_melhor $\leftarrow 1$
                        Pare;
                Se !existe_melhor então:
                    $Q^x \leftarrow Q^x \cup \{Q_k\}$;
                    $Q^A \leftarrow Q^A \cup \{Q_k\}$;

que a de corte, em alguma consulta retirada de Q^A, de forma que estas consultas sejam então associadas a essa coleção.

No passo (iv), se o número de consultas que são favorecidas pelas coleções de dimensão é menor que 80% das consultas ($|Q^A|/|Q| < 0,8$), a coleção com modelagem de agregado baseada na tabela de fato C^F é adicionada ao conjunto R.

No passo (v), as consultas para as quais não tenha sido definido qual a coleção que melhor as favorecem (consultas órfãs), devem ser associadas à alguma coleção. Se existir a coleção baseada na tabela de fato no conjunto R, tais consultas serão associadas a esta coleção. Caso contrário, estas consultas serão associadas à coleção baseada em uma tabela de dimensão que possua a menor SCR de cada consulta órfã ou à que possua o maior número de documentos se a SCR das coleções em R forem iguais.

Para exemplificar o uso da heurística proposta iremos aplicá-la para a escolha das modelagens de agregados que podem ser utilizadas para o conjunto de consultas do SSB. Para isso utilizaremos a seletividade de corte de 0,1 para todas as modelagens de agregado possíveis a partir do esquema do TPC-DS e as SCR da tabela 4.3:

No passo (i) o conjunto R será definido como vazio.

No passo (ii) os seguintes conjuntos serão formados:

A coleção *Orderdate* possui a SCR menor que a de corte nas consultas 1.2, 1.3 e 3.4, porém somente nas consultas 1.2 e 1.3 estas SCR são menores que as das outras coleções

Logo:

- $Q_L^{Orderdate} = \{1.2, 1.3, 3.4\}$.
- $Q^{Orderdate} = \{1.2, 1.3\}$.
- $R = \{C^{Orderdate}\}$.
- $Q^A = \{1.2, 1.3\}$.

A coleção *Part* possui a SCR menor que a de corte nas consultas 2.1, 2.2, 2.3 e 4.3.

Logo:

- $Q_L^{Part} = \{2.1, 2.2, 2.3, 4.3\}$.
- $Q^{Part} = \{2.1, 2.2, 2.3, 4.3\}$
- $R = \{C^{Orderdate}, C^{Part}\}$
- $Q^A = \{1.2, 1.3, 2.1, 2.2, 2.3, 4.3\}$

Ainda no passo (ii), devido a coleção *Customer* possuir um número maior de documentos que a coleção *Supplier* e possuir nas consultas 3.2 a 3.4 a mesma SCR, menor que a de corte, estas consultas serão adicionadas aos conjuntos $Q_L^{Customer}$ e $Q^{Customer}$ e esta coleção também será adicionada ao conjunto R.

Logo:

- $Q_L^{Customer} = \{3.2, 3.2, 3.4\}$.
- $Q^{Customer} = \{3.2, 3.3, 3.4\}$.
- $R = \{C^{Orderdate}, C^{Part}, C^{Customer}\}$.
- $Q^A = \{1.2, 1.3, 2.1, 2.2, 2.3, 4.3, 3.2, 3.3, 3.4\}$.

A coleção *Supplier* possui a menor SCR nas consultas 3.2, 3.3 e 3.4, junto com outras coleções, porém as outras possuem um número maior de documentos nessas consultas, logo *Supplier* não será adicionada a R e os conjuntos R e Q^A não serão alterados:

- $Q_L^{Supplier} = \{3.2, 3.3, 3.4, 4.3\}$.
- $Q^{Supplier} = \{\}$.
- $R = \{C^{Orderdate}, C^{Part}, C^{Customer}\}$.
- $Q^A = \{1.2, 1.3, 2.1, 2.2, 2.3, 4.3, 3.2, 3.3, 3.4\}$.

No início do passo (iii) haverá a ordenação do conjunto R. Como as coleções *Orderdate* e *Customer* possuem a SCR menor que a de corte em um mesmo número de consultas, mas a coleção *Orderdate* favorece um número menor de consultas essa será a primeira do conjunto R, logo:

- $R = \{C^{Orderdate}, C^{Customer}, C^{Part}\}$.

Como a coleção *Orderdate* é a primeira no conjunto R ordenado e $|Q^{Orderdate}| < 0,2$ esta coleção será retirada de R e as consultas de $Q^{Orderdate}$ serão retiradas de Q^A. Como todas as outras coleções de dimensão não possuem a SCR menor que a de corte nessas consultas, nenhuma delas deverá executar essas consultas. Depois da coleção *Orderdate* nenhuma outra existente em R favorece menos de 20% das consultas, então somente esta será excluída de R.

Logo:

- $R = \{C^{Part}, C^{Customer}\}$.
- $Q^A = \{2.1, 2.2, 2.3, 4.3, 3.2, 3.3, 3.4\}$.

No passo (iv), como $|Q^A| = 7$ e $|Q| = 13$, $|Q^A|/|Q| < 0,8$, logo:

- $R = \{C^{Part}, C^{Customer}, C^{Lineorder}\}$.

No passo (v), como $C^F \in R$ e $Q - Q^A = \{1.1, 1.2, 1.3, 3.1, 4.1, 4.2\}$ (consultas órfãs):

- $Q^{C^F} = \{1.1, 1.2, 1.3, 3.1, 4.1, 4.2\}$.

Ao final, o conjunto de coleções escolhido é formado pelas coleções *Part*, *Customer* e *Lineorder*. Neste conjunto, sabe-se que cada coleção atende melhor um subconjunto das consultas usadas como base da heurística. No entanto, algumas consultas podem ficar de fora. No caso do *benchmark* SSB, somente as consultas 1.2 e 1.3 não poderiam ser executadas sobre a(s) coleção(ões) que mais as favoreceriam (coleção *Orderdate*), devido ao descarte desta coleção. Dentro do conjunto de coleções selecionadas, estas consultas seriam melhores atendidas pela coleção *Lineorder*, onde o ganho médio final no desempenho das consultas do SSB obtido pela utilização da heurística seria de 27%.

4.7 RESULTADOS E HEURÍSTICAS NO ELASTICSEARCH

Na tabela 4.4 podemos observar os tempos de respostas das consultas realizadas sobre o Elasticsearch nas diferentes coleções com diferentes modelagens de agregado e, da mesma forma que no MongoDB, podemos verificar que algumas consultas foram favorecidas por certas modelagens de agregados.

Analisando os resultados do ES na tabela 4.4, é possível observar que o comportamento do ES foi diferente do MongoDB. A primeira diferença que pode ser verificada é na coleção que obteve a menor média de tempo nas consultas. No MongoDB, a coleção *Lineorder* foi a que obteve a menor média de tempo. Já no ES, a coleção que obteve a menor média foi a coleção *Supplier*, enquanto que a coleção *Lineorder* foi a que obteve a pior média entre todas as coleções.

O comportamento observado no ElasticSearch está relacionado à presença de um campo raiz da coleção, no filtro das consultas. Em todas as consultas, se um campo raiz da coleção baseada na dimensão estava presente no filtro da consulta, isto foi suficiente para que a coleção obtivesse um tempo de resposta melhor do que a coleção baseada na tabela de fato (*Lineorder*).

TAB. 4.4: Tempo médio das consultas nas diferentes coleções no ES em milissegundos.

Sequential (ms)	Lineorder	Customer	Orderdate	Part	Supplier	Gain
1.1	159	172	93	186	155	41
1.2	110	37	21	35	69	81
1.3	107	30	17	33	48	84
2.1	103	72	73	79	32	69
2.2	107	59	81	33	31	71
2.3	96	23	36	15	25	84
3.1	142	74	77	160	64	55
3.2	109	35	65	62	24	77
3.3	104	20	24	43	17	83
3.4	86	18	14	24	14	84
4.1	150	72	102	107	67	55
4.2	122	52	48	110	45	63
4.3	100	27	29	38	22	77
Média	115	53	52	71	47	69

No entanto, foi possível notar que a seletividade (SCR) dos filtros das consultas não permitiu distinguir sobre quais coleções dimensão essas consultas teriam o melhor desempenho.

Por exemplo, a consulta 2.2 utiliza como filtro um campo raiz da coleção *Part* e um campo raiz da coleção *Suppplier*. Como pode ser visto na tabela 4.5, para a coleção *Part* a SCR é 0,008, enquanto que para a coleção *Supplier*, a SCR é 0,2. Embora a coleção *Part* tenha uma SCR menor, esta consulta teve um pior desempenho em relação à coleção *Supplier*.

TAB. 4.5: Seletividade dos campos raízes das coleções do SSB

Consultas	Lineorder	Customer	Orderdate	Part	Supplier
1.1	0,166		0,142		
1.2	0,045		0,011		
1.3	0,045		0,002		
2.1				0,04	0,2
2.2				0,008	0,2
2.3				0,001	0,2
3.1		0,2	0,875		0,2
3.2		0,04	0,875		0,04
3.3		0,008	0,875		0,008
3.4		0,008	0,011		0,008
4.1		0,2		0,4	0,2
4.2		0,2	0,25	0,4	0,2
4.3		0,2	0,25	0,008	0,04

Assim, baseado nos resultados dos experimentos, estabeleceu-se uma heurística para

o ES que permite a escolha das coleções baseada somente na presença do campo raiz como filtro das consultas.

Como o principal fator de influência do comportamento do ES foi o fato de uma coleção possuir um campo raiz na consulta, a sua heurística selecionará inicialmente a coleção que possui seus campos raiz no maior número de consultas e associará essas consultas a essa coleção. Entretanto, a primeira coleção escolhida poderá não ter campos raízes em todas as consultas existentes, logo é necessário escolher outra coleção somente para as consultas órfãs, utilizando o mesmo critério.

Além disso, a heurística proposta deve prever a existência de consultas que não utilizem nenhum campo das raízes das coleções baseadas nas tabelas de dimensão, nas cláusulas de seleção, embora isso não ocorra nas consultas dos experimentos deste trabalho.

Sejam:

- $\phi_{Q_n}(x)$ um termo booleano, onde seu valor verdadeiro representa o fato de que uma coleção $x \in C^D$ possui um campo raiz na consulta Q_n.
- Q_D o conjunto de consultas que possuem pelo menos um campo das suas cláusulas de seleção, pertencente à raiz de uma coleção de C^D.
- Q_r^x o conjunto de consultas em que uma coleção $x \in C^D$ possui pelo menos um campo raiz nas suas cláusulas de seleção.
- Ordena(R) função que ordena o conjunto R de acordo com o número de consultas associadas a cada coleção, onde o primeiro elemento após a ordenação será a coleção que possui o menor número de consultas associadas.

Portanto, a formalização da heurística de escolha de modelagens de agregados do ElasticSearch pode ser vista no algoritmo 2.

Da mesma forma feita para explicação da heurística do MongoDB, o algorítmo da heurística do ElasticSearch foi dividido em passos para facilitar a sua explicação.

O passo (i) da heurística atribui vazio aos conjuntos R e Q^A.

No passo (ii), as consultas que não possuem nenhum campo pertencente às raízes das coleções baseadas nas tabelas de dimensão, nas suas cláusulas de seleção, devem ser tratadas de forma diferente das demais, pois o critério de escolha das coleções se baseia no número de consultas em que cada uma possui campos raízes. Logo essas consultas não poderiam ser utilizadas nos passos seguintes da heurística. No final deste passo, se o número dessas consultas for maior que 20% do total de consultas, a coleção de baseada na tabela de fatos será adicionada ao conjunto R e associará essas consultas a ela.

Algorithm 2 Heurística do ElasticSearch (E1).

Entrada: C^D, Q, $\phi_{Q_n}(C^{D_i})$, C^F
Saida: R; Q^x,$\forall x \in C^D$; Q^{C^F}
função seleciona_modelagens(C^D, Q, $\phi_{Q_n}(C^{D_i})$, C^F)
Passo(i):
    $R \leftarrow \{\}$; $Q^A \leftarrow \{\}$;
Passo(ii):
    Para cada $Q_k \in Q$, faça:
        Para cada $x \in C^D$, faça:
            Se $\phi_{Q_k}(x)$ então:
                $Q_D \leftarrow Q_D \cup \{Q_k\}$;
    Se $|Q - Q_D|/|Q| > 0,2$ então:
        $Q^{C^F} \leftarrow Q - Q_D$; $Q^A \leftarrow Q - Q_D$; $R \leftarrow R \cup \{C^F\ \}$;
Passo(iii):
    Enquanto $Q_D \subsetneq Q_A$, faça: % Até que todas as consultas de Q_D sejam atendidas por alguma coleção
        Para cada $x \in \{C^D - R\}$, faça:
            $Q_r^x \leftarrow \{\}$;
            Para cada $Q_k \in \{Q_D - Q^A\}$, faça:
                Se $\phi_{Q_k}(x)$ então:
                    $Q_r^x \leftarrow Q_r^x \cup \{Q_k\}$;
        Para cada $x \in \{C^D - R\}$, faça:
            existe_melhor $\leftarrow 0$;
            Para cada $y \in \{C^D - R\}$, onde y $\neq$ x, faça:
                Se $|Q_r^y| > |Q_r^x|$ então:
                    existe_melhor $\leftarrow 1$;
                    Pare;
            Se !existe_melhor então:
                $Q^A \leftarrow Q^A \cup Q_r^x$; $Q^x \leftarrow Q^x \cup Q_r^x$; $R \leftarrow R \cup \{x\}$;
Passo(iv):
    $R \leftarrow$ Ordena(R);
    Para cada $x \in R$, faça:
        Se $|Q^x|/|Q| < 0,2$ então:
            $R \leftarrow R - \{x\}$;
            $Q^A \leftarrow Q^A - \{Q^x\}$;
Passo(v):
    Se $|Q^A| \neq |Q|$ então:
        Para cada $x \in R$, faça:
            existe_melhor $\leftarrow 0$;
            Para cada $y \in R$, onde y $\neq$ x, faça:
                Se $|Q^y| > |Q^x|$ então:
                    existe_melhor $\leftarrow 1$;
                    Pare;
            Se !existe_melhor então:
                $Q^x \leftarrow Q^x \cup (Q - Q^A)$;

Para as demais consultas, será criado um conjunto Q_D que definirá aquelas que realmente influenciarão na escolha entre as coleções baseadas nas tabelas de dimensão.

No passo (iii) serão escolhidas as coleções baseadas nas tabelas de dimensão que possuem campos raízes na cláusula de seleção, no maior número de consultas que ainda não formam associadas, até que todas as consultas sejam associadas a alguma coleção. Como na primeira execução ainda não foi associada nenhuma consulta, é escolhida a coleção que possui campos raízes no maior número de consultas, levando em consideração todas as consultas.

Nas iterações seguintes, se ainda houver consulta não associada, a escolha deve se basear somente nessas consultas. Por esse motivo, a cada iteração dentro do laço enquanto, o conjunto Q_r^x é instanciado com o conjunto vazio, de forma que a quantidade de consultas que possuem campos raízes de cada coleção, em cada iteração, considere somente as consultas que ainda não foram associadas $(Q_D - Q^A)$.

No final do laço enquanto, ao escolhermos uma coleção x, as consultas de Q_r^x serão associadas a x $(Q^x \leftarrow Q^x \cup Q_r^x)$ e adicionadas a Q^A, sinalizando que estas consultas não são mais órfãs.

No passo (iv), após a ordenação do conjunto R, são descartadas as coleções que possuem menos de 20% das consultas associadas a elas. Ao retirar uma coleção x do conjunto R, as consultas que estavam associadas ao conjunto retirado Q^x passarão a ser consultas órfãs, pois como o conjunto R foi ordenado segundo o número de consultas associadas a cada coleção, a coleção descartada foi a última a ser adicionada a R no passo anterior e as consultas associadas a x não possuem campos raízes nas outras coleções de R.

No passo (v), as consultas órfãs serão associadas à coleção que possui o maior número de consultas associadas.

Seguindo a heurística do ES e utilizando a tabela 4.6, onde as coleções que possuem um campo raiz na cláusula de seleção das consultas do SSB estão marcadas com um "X", é possível selecionar as modelagens de agregado que melhor favorecem essas consultas.

No passo (i) da heurística seria atribuído vazio aos conjuntos R e Q^A.

No passo (ii) como todas as consultas do SSB possuíam pelo menos um campo raiz nas suas cláusulas de seleção, os conjuntos R, Q^A e Q^{C^F} permaneceram vazios e $Q_D = Q$.

No início do passo (iii) os seguintes conjuntos foram formados:

- $Q^{Customer} = \{3.1, ..., 4.3\}$, $|Q^{Customer}| = 7$;
- $Q^{Orderdate} = \{1.1, 1.2, 1.3, 3.1, ..., 3.4, 4.2, 4.2\}$, $|Q^{Orderdate}| = 9$;

- $Q^{Part} = \{2.1, 2.2, 2.3, 4.1, 4.2, 4.3\}$, $|Q^{Part}| = 6$; e
- $Q^{Supplier} = \{2.1, ..., 4.3\}$, $|Q^{Supplier}| = 10$;

A coleção *Supplier* possui campos raízes nas consultas 2.1 a 4.3 e como esta coleção possui campos raízes na cláusula de seleção do maior número de consultas ($|Q_R^{supplier}| = 10$) esta deve ser adicionada a R, logo;

- $R = \{C^{Supplier}\}$
- $Q^{Supplier} = \{2.1, ..., 4.3\}$
- $Q^A = \{2.1, ..., 4.3\}$

Ainda no passo (iii) como Q^A ainda é diferente de Q_D. Deve-se executar o laço novamente para as consultas 1.1 a 1.3. Como a coleção *Orderdate* é a única que possui campos raízes nas cláusulas de seleção das consultas de 1.1 a 1.3, esta deve ser adicionada a R, logo:

- $R = \{C^{Supplier}, C^{Ordedate}\}$
- $Q^{Orderdate} = \{1.1, 1.2, 1.3\}$
- $Q^A = \{1.1, ..., 4.3\}$

Como neste ponto $Q^A = Q_D$ passaremos para o passo (iV).

No passo (iv), como todas as coleções em R possuem associadas as essas mais de 20% das consultas, não é necessário descartar nenhuma coleção.

No passo (v), como todas as consultas estão associadas a alguma coleção ($Q = Q^A$), este passo não será executado.

Portanto, ao final da aplicação da heurística teremos os seguintes conjuntos:

- $R = \{C^{Supplier}, C^{Ordedate}\}$
- $Q^{Supplier} = \{2.1, ..., 4.3\}$
- $Q^{Orderdate} = \{1.1, 1.2, 1.3\}$
- $Q^A = \{1.1, ..., 4.3\}$

Ao final do algoritmo, todas as consultas são atendidas por alguma coleção. Vale notar que somente a consulta 2.3 não foi associada à coleção que mais a favoreceu. O ganho médio final obtido após a utilização da heurística foi de 68% se compararmos os tempos das consultas sobre as coleções escolhidas com os tempos das consultas sobre a coleção *Lineorder*.

TAB. 4.6: Coleções que possuem campos raízes em cada consulta do SSB.

Consultas	Lineorder	Customer	Orderdate	Part	Supplier
1.1	X		X		
1.2	X		X		
1.3	X		X		
2.1				X	X
2.2				X	X
2.3				X	X
3.1		X	X		X
3.2		X	X		X
3.3		X	X		X
3.4		X	X		X
4.1		X		X	X
4.2		X	X	X	X
4.3		X	X	X	X

4.8 RESULTADOS E HEURÍSTICAS NO ORIENTDB

O OrientDB permite que os dados no formato JSON sejam organizados de duas formas distintas. A primeira forma é mais similiar à forma utilizada pelos outros SGBD orientados a documentos, onde um documento em JSON é um documento único, com dados aninhados. Neste trabalho, esta forma de organizar os dados será intitulada como documentos aninhados. A outra forma é uma forma conhecida como *linked* (documentos ligados), onde os campos aninhados são tratados como documentos diferentes, porém os campos aninhados estão ligados através de um mapeamento aos documentos raízes.

4.8.1 RESULTADOS E HEURÍSTICAS NO ORIENTDB UTILIZANDO DOCUMENTOS ANINHADOS

A partir dos resultados do OrientDB, quando utilizando documentos aninhados, na tabela 4.7, e a partir da tabela 4.8 que apresenta a SCR das coleções nas consultas do SSB, é possível observar que as coleções em que as consultas obtiveram um melhor desempenho, foram as que tinham a menor SCR em cada consulta, da mesma forma que no MongoDB, e que quando todas as coleções baseadas nas tabelas de dimensão tinham um valor de SCR acima de 0,04, a coleção que mais favoreceu a consulta foi a coleção *Lineorder*.

Entretanto diferentemente do MongoDB, nas consultas do SSB, onde o valor de SCR das coleções eram maiores ou iguais que 0,2 ou quando a coleção não possuía algum campo raiz na cláusula de seleção das consultas, um erro de execução no OrientDB conhecido como *Garbage Collector Overhead Limit Exceed* (GCOLE) ocorreu nas coleções baseadas

TAB. 4.7: Tempo de médio, em milissegundos, das consultas nas diversas coleções no OrientDB, utilizando documentos aninhados.

Consulta	Lineorder	Customer	Orderdate	Part	Supplier	Ganho
1.1	247660					0%
1.2	247662		115556			53%
1.3	242383		161858			33%
2.1	225112			177136		21%
2.2	296162			172096		41%
2.3	222916			171903		22%
3.1	243989					0%
3.2	238453	142133			280282	40%
3.3	242087	181583			372003	24%
3.4	241580	183799	113509		368419	53%
4.1	241707					0%
4.2	240495					0%
4.3	239301			177354	266724	25%
Média	243808	169171	130307	174622	321857	24%

TAB. 4.8: Seletividade dos campos raízes das coleções do SSB, e no final, o número de documentos em cada coleção de agregado

Consultas	Lineorder	Customer	Orderdate	Part	Supplier
1.1	0,166		0,142		
1.2	0,045		0,011		
1.3	0,045		0,002		
2.1				0,04	0,2
2.2				0,008	0,2
2.3				0,001	0,2
3.1		0,2	0,875		0,2
3.2		0,04	0,875		0,04
3.3		0,008	0,875		0,008
3.4		0,008	0,011		0,008
4.1		0,2		0,4	0,2
4.2		0,2	0,25	0,4	0,2
4.3		0,2	0,25	0,008	0,04
Documentos	6001215	30000	2556	200000	2000

nas tabelas de dimensão, não sendo possível medir o tempo de resposta dessas consultas.

Este erro ocorre em aplicações Java quando a máquina virtual Java (JVM) está gastando muito tempo fazendo o *Garbage Collector*(GC), ou seja, está limpando a memória *heap* do Java, porém com pouco resultado. Por padrão, a JVM é configurada para lançar este erro quando a JVM gasta 98% do tempo realizando o GC e apenas 2% da memória *heap* é recuperada[21]. Fez-se uma tentativa de desabilitar este erro no OrientDB, porém,

[21]https://plumbr.eu/outofmemoryerror/gc-overhead-limit-exceeded

ao fazer isso, as consultas não finalizaram.

Além dessa, várias configurações do OrientDB foram testadas visando evitar este erro, porém na maioria das consultas, quando este erro era evitado, o erro de falta memória *heap* ocorria. Assim, a ocorrência deste erro foi considerada como uma dificuldade do SGBD em lidar com documentos que possuem uma quantidade elevada de dados aninhados e que necessitam da realização da operação *unwind* para que as consultas sejam respondidas.

Este fato pode ser confirmado pelo fato de que todas as consultas na coleção *Lineorder* foram executadas, sem a ocorrência desse erro, e o motivo para isto foi a ausência da necessidade de realizar a operação *unwind* nesta coleção para as consultas realizadas.

De forma similar ao MongoDB, foram realizados testes, visando verificar a existência um valor de corte para a SCR das coleções baseadas nas tabelas de dimensão, onde para SCR abaixo desse valor essas coleções terão melhor desempenho do que a coleção baseada na tabela *Lineorder*. A partir da figura 4.23, é possível observar que, em todas as coleções, foi encontrado um valor para a seletividade de corte entre 0,04 e 0,05. Assim, com base nessa observação, foi escolhido o valor de corte de 0,05 para todas as coleções.

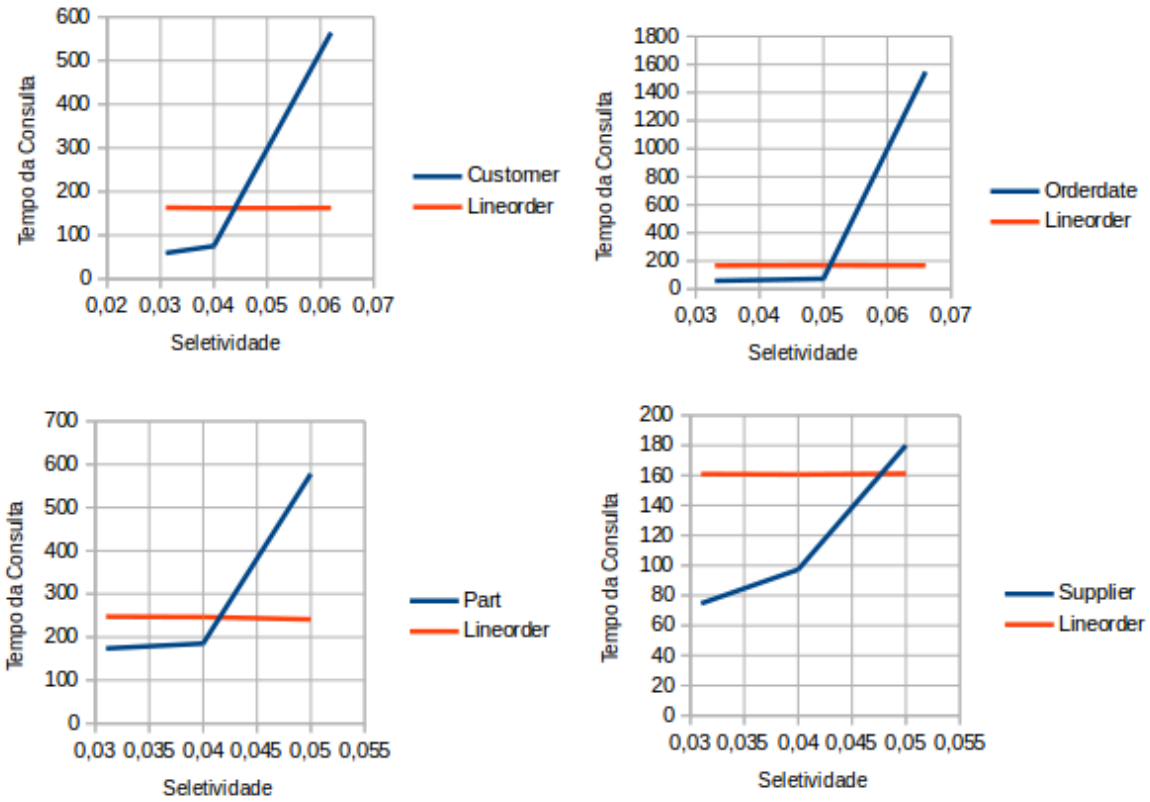

FIG. 4.23: Gráficos Tempo x Seletividade no OrientDB utilizando documentos aninhados.

Outro comportamento do OrientDB, utilizando documentos aninhados, similar ao do

MongoDB, refere-se ao fato de que entre duas coleções que possuíam SCR equivalentes, a coleção que possuía o maior número de documentos favoreceu mais a consulta. Portanto, da mesma forma que no MongoDB, será considerado que o maior número de documentos de uma coleção pesará na escolha quando uma coleção possuir o mesmo SCR que outra em uma consulta.

Devido ao comportamento similar desses dois SGBD, a heurística para escolha de modelagens agregados para o OrientDB utilizando documentos aninhados é similar à heurística do MongoDB. A diferença existe com relação à inclusão da modelagem baseada na tabela de fato, pois as consultas que não são favorecidas pelas coleções de dimensão podem não ser realizadas devido ao erro GCOLE, logo para que seja possível a execução de novas consultas cuja a SCR das coleções de dimensão escolhidas venham a ser muito altas, na heurística do OrientDB utilizando dados aninhados, a coleção baseada na tabela de fatos será construída sempre.

A formalização da heurística do OrientDB utilizando documentos aninhados, é apresentada no algoritmo 3. Foram utilizadas as mesmas definições estabelecidas por ocasião da formalização da heurística do MongoDB.

Como a heurística do OrientDB é muito similar à do MongoDB explicaremos somente as pequenas diferenças nos passos que diferenciam ambas as heurísticas.

A primeira diferença, como comentado anteriormente ocorre no passo (i), pois na heurística do OrientDB o conjunto R é inicializado com o elemento que representa a coleção baseada na tabela de fatos.

Os passos (ii) e (iii) do OrientDB são iguais aos do MongoDB.

Como o passo (iv) da heurística do MongoDB não faz sentido no OrientDB, este foi substituído por um laço que associa as consultas que não foram associadas a nenhuma coleção nos passos (ii) e (iii) (consultas órfãs), à coleção baseada na tabela de fatos.

Como nas consultas do SSB, nenhuma coleção possui a SCR entre os valores de 0,05 e 0,1, correspondente à seletividade de corte do OrientDB e do MongoDB, as coleções selecionadas a partir de ambas as heurísticas serão as mesmas, onde a única diferença na aplicação da heurística do OrientDB ocorre no momento em que a coleção baseada na tabela de fato é incluída. Portanto, não demostraremos a aplicação da heurística do OrientDB utilizando documentos aninhados para as consultas e coleções do SSB.

Apesar de as modelagens de agregados escolhidas serem as mesmas após a utilização das heurísticas no MongoDB e no OrientDB o ganho obtido no OrientDB foi um pouco menor (19%).

Algorithm 3 Heurística do OrientDB documentos aninhados (OA1).

Entrada: C^D, $\eta(C^{D_i})$, Q, $\sigma_{Q_n}(C^{D_i})$, C^F, $L^{C^{D_i}}$
Saída: R; Q^x,$\forall x \in C^D$; Q^{C^F}
função seleciona_modelagens(C^D, C^F, $\eta(C^{D_i})$, Q, $\sigma_{Q_n}(C^{D_i})$, $L^{C^{D_i}}$)
Passo(i):
    $R \leftarrow \{C^F\}$;
Passo(ii):
    Para cada $x \in C^D$, faça:
        Para cada $Q_k \in Q$, faça:
            Se $\sigma_{Q_k}(x) < L^x$ então:
                $Q^x_L \leftarrow Q^x_L \cup \{Q_k\}$;
                existe_melhor $\leftarrow 0$
                Para cada $y \in C^D$, onde $y \neq x$, faça:
                    Se ($\sigma_{Q_k}(y) < \sigma_{Q_k}(x)$) ou ($\sigma_{Q_k}(y) = \sigma_{Q_k}(x)$ e $\eta(y) > \eta(x)$) então:
                        existe_melhor $\leftarrow 1$
                        Pare;
                Se !existe_melhor então:
                    $R \leftarrow R \cup \{x\}$;
                    $Q^x \leftarrow Q^x \cup \{Q_k\}$;
                    $Q^A \leftarrow Q^A \cup \{Q_k\}$;
Passo(iii):
    $R \leftarrow$ Ordena(R);
    Para cada $x \in R$, faça:
        Se $|Q^x|/|Q| < 0,2$ então:
            $R \leftarrow R - \{x\}$
            $Q^A \leftarrow Q^A - Q^x$
            Para cada $z \in R$, faça:
                Para cada $Q_k \in Q^x$, faça:
                    Se $\sigma_{Q_k}(z) < L^z$ então:
                        existe_melhor $\leftarrow 0$;
                        Para cada $y \in R$, onde $y \neq z$, faça:
                            Se ($\sigma_{Q_k}(y) < \sigma_{Q_k}(z)$) ou ($\sigma_{Q_k}(y) = \sigma_{Q_k}(z)$ e $\eta(y) > \eta(z)$) então:
                                existe_melhor $\leftarrow 1$;
                                Pare;
                        Se !existe_melhor então:
                            $Q^z \leftarrow Q^z \cup \{Q_k\}$;
                            $Q^A \leftarrow Q^A \cup \{Q_k\}$;
Passo(iv):
    $Q^{C^F} \leftarrow Q - Q^A$;

4.8.2 RESULTADOS E HEURÍSTICAS NO ORIENTDB UTILIZANDO DOCUMENTOS LIGADOS

Da mesma forma que nos resultados do MongoDB e do OrientDB utilizando documentos aninhados, nos resultados do OrientDB utilizando documentos ligados, ao analisarmos as SCR das coleções nas consultas, por meio da tabela 4.10 e os tempos das consultas mostrados na tabela 4.9, podemos notar que a SCR influenciou significantemente no desempenho das consultas sobre as coleções baseadas nas tabelas dimensão. Para os valores mais baixos, as coleções baseadas nas tabelas de dimensão tiveram um melhor desempenho que a coleção baseada na tabela de fato. O motivo provável para este comportamento em comum, nestes SGBD, foi a necessidade de realizar a operação de *unwind* nas coleções baseadas nas tabelas de dimensão. Portanto, da mesma forma realizada em experimentos anteriores, buscou-se encontrar uma seletividade de corte para esta forma de utilizar o OrientDB.

TAB. 4.9: Tempo de médio, em milissegundos, das consultas nas diversas coleções no OrientDB, em documentos ligados.

Consulta	Lineorder	Customer	Orderdate	Part	Supplier	Ganho
1.1	115065	254624[a]	79320	105692[a]	231593[a]	31%
1.2	109585	254229[a]	7184	100099[a]	231212[a]	93%
1.3	104845	243459[a]	1827	93868[a]	222342[a]	98%
2.1	256827	196428[a]	157636[b]	12442	142280	95%
2.2	292375	229432[a]	257326[b]	4706	211061	98%
2.3	128392	197348[a]	150312[b]	2302	145043	98%
3.1	270995	121549	199431[b]	201122[a]	244953	55%
3.2	235080	16527	192897[b]	195167[a]	28482	92%
3.3	158671	2682	196788[b]	145304[a]	5249	98%
3.4	158572	2586	8145	145856[a]	5356	98%
4.1	290704	135515	195335[b]	231318[b]	185027	53%
4.2	287319	92681	320401	224299[b]	173991	67%
4.3	258380	90964	335208	12431	30367	95%
Média	205139	141386	161677	113431	142842	82%

A partir dos gráficos da figura 4.24 pode-se observar que para a coleção *Orderdate* o valor da seletividade de corte foi próxima a 0,23 e para a coleção *Supplier* esta foi próxima a 0,21. Entretanto, para as outras coleções baseadas nas tabelas de dimensão, este valor não foi encontrado, pois ao utilizar um valor da SCR maior que 0,2 nas consultas sobre as coleções *Customer* e *Part*, o erro GCOLE ocorreu.

Nos experimentos iniciais com o OrientDB, foram realizadas tentativas de executar as consultas em todas as coleções, mesmo quando estas possuíam uma SCR alta, porém o

TAB. 4.10: Seletividade dos campos raízes das coleções do SSB, e no final, o número de documentos em cada coleção de agregado

Consultas	Lineorder	Customer	Orderdate	Part	Supplier
1.1	0,166		0,142		
1.2	0,045		0,011		
1.3	0,045		0,002		
2.1				0,04	0,2
2.2				0,008	0,2
2.3				0,001	0,2
3.1		0,2	0,875		0,2
3.2		0,04	0,875		0,04
3.3		0,008	0,875		0,008
3.4		0,008	0,011		0,008
4.1		0,2		0,4	0,2
4.2		0,2	0,25	0,4	0,2
4.3		0,2	0,25	0,008	0,04
Documentos	6001215	30000	2556	200000	2000

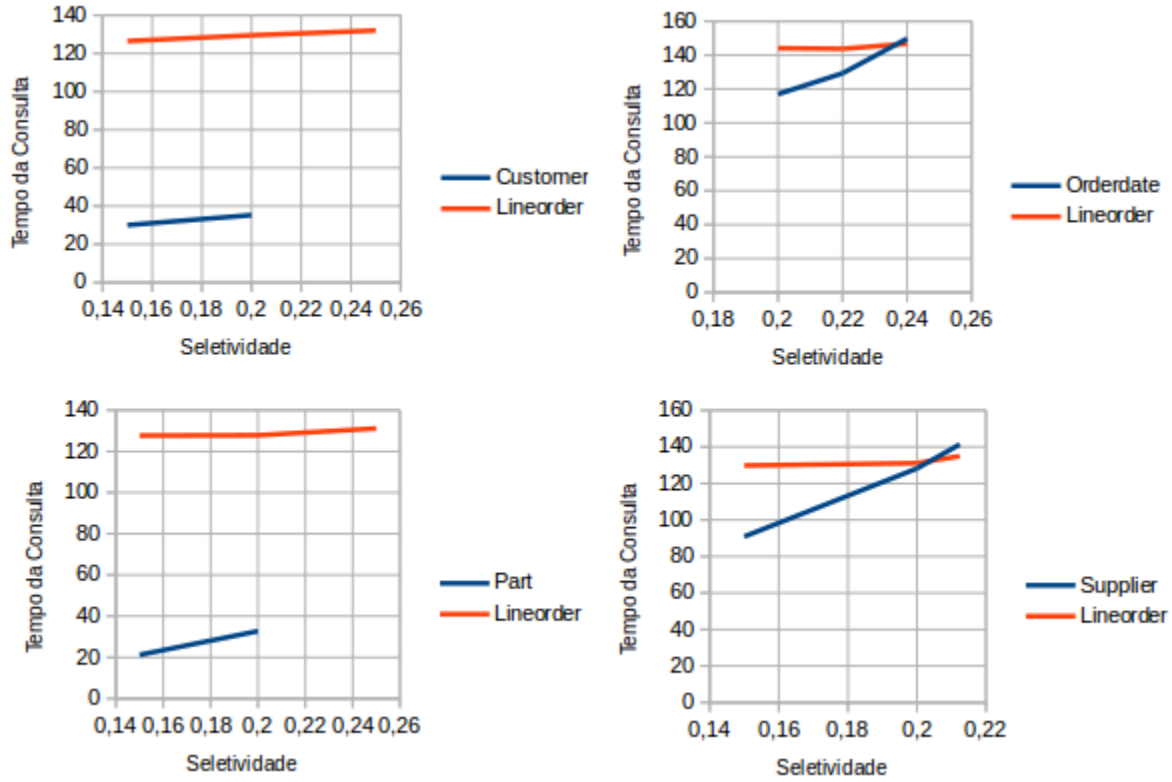

FIG. 4.24: Gráficos Tempo x Seletividade no OrientDB utilizando documentos ligados.

erro GCOLE impedia que essas consultas fossem realizadas. E nas consultas onde a SCR era menor ou igual a 0,2, as coleções baseadas nas tabelas de dimensão tiveram melhor desempenho.

Portanto, apesar de não ter sido possível obter um valor de seletividade de corte real para as coleções *Customer* e *Part*, o valor de 0,2 pode ser utilizado para a seletividade de corte.

Além deste comportamento acima, foi observado outro comportamento comum aos dois experimentos já citados anteriormente. Ao realizar uma consulta em duas coleções diferentes com a mesma SCR a coleção com um maior número de documentos obteve um melhor desempenho. Este comportamento pode ser observado nas consultas 3.1 a 4.2 da tabela 4.9, onde a coleção *Customer* obteve melhor desempenho que a coleção *Supplier* em todas essas consultas, pois aquela possui um número maior de documentos que esta e a SCR dessas coleções eram iguais nessas consultas. Portanto, para SCR iguais deve-se priorizar a coleção com um maior número de documentos.

Devido à forma de organização dos dados no OrientDB, ao utilizar documentos ligados, (já explicado na seção 2.2.3) é possível realizar consultas diretamente sobre a classe de dados aninhados, utilizando ou não relacionamentos reversos, conforme explicado na seção 4.3.

Tendo em vista, que algumas consultas não puderam ser realizadas, devido às coleções possuírem a SCR mais alta que 0,2, foram realizados testes de forma a verificar a possibilidade da execução dessas consultas diretamente na classe de dados aninhados e analisar o desempenho quando realizando as consultas desta forma.

Na seção 4.3 foi demonstrado que quando uma consulta é realizada diretamente na classe de dados aninhados em uma coleção que não possui nenhum campo raiz nas cláusulas de seleção ou de projeção, esta pode ser escrita de forma que não necessite da utilização de relacionamentos reversos.

Na tabela 4.9 os tempos das consultas que foram realizadas desta forma estão sinalizadas com a letra ”a” sobrescrita.

A consulta 2.1, é um exemplo de consulta realizada sobre a coleção *Customer* diretamente na classe de dados aninhados que não precisou utilizar relacionamentos reversos. Nesta consulta seu desempenho foi melhor do que a mesma consulta realizada na coleção *Lineorder*. Em outras consultas, como a 2.3, a coleção *Lineorder* teve melhor desempenho do que a mesma consulta realizada na coleção *Customer* diretamente sobre a classe de dados aninhados. Entretanto, ao compararmos todos os tempos das consultas sobre as coleções de dimensão, realizadas diretamente na classe de dados aninhada, sem utilizar relacionamentos reversos, com os tempos das mesmas consultas sobre a coleção *Lineorder*, podemos notar que a maioria dos tempos da primeira forma foram melhores.

Logo, será considerado que de maneira geral, as consultas realizadas sobre as coleções

baseadas nas tabelas de dimensão, diretamente na classe de dados aninhados, apresentam um melhor desempenho do que as consultas realizadas sobre a coleção baseada na tabela de fatos.

Além das consultas diretamente sobre a classe de dados aninhados, sem utilizar relacionamentos reversos, nas consultas sobre as coleções que possuíam uma SCR maior que a de corte ou que possuíam campos raízes somente na cláusula de projeção, foi necessário criar relacionamentos reversos, de forma que estas pudessem ser executadas sem que ocorresse o erro GCOLE.

Na tabela 4.9 os tempos das consultas que utilizaram relacionamento reverso estão sinalizados com a letra ”b” sobrescrita.

Da mesma forma que as consultas diretamente sobre a classe de dados aninhados, sem utilizar relacionamentos, as consultas utilizando relacionamentos reversos também obtiveram um melhor desempenho do que a coleção baseada na tabela de fatos na maioria das consultas. Logo, também pode-se considerar que realizar as consultas desta forma é mais eficiente do que realizar a consulta sobre a coleção baseada na tabela de fato.

Portanto, diferentemente do OrientDB utilizando dados aninhados, para resolver as consultas sobre as coleções que não possuem os campos raízes na cláusula de seleção das consultas, ou cuja SCR seja mais alta do que a de corte, não é necessário manter a coleção baseada na tabela de fatos, basta realizá-las sobre a classe de dados aninhados de qualquer coleção de dimensão utilizando ou não relacionamento.

Por vezes, uma consulta que é realizada sobre coleção diretamente na classe de dados aninhados sem a utilização de relacionamentos reverso, pode vir a precisar deste recurso quando esta é realizada sobre outra coleção. Este fato poderia influenciar na escolha, entre uma coleção ou outra, se dentre estas duas formas de realizar consultas, uma fosse mais eficiente que a outra. Porém, a partir dos experimentos, não foi possível concluir qual das duas formas permite um melhor desempenho das consultas.

Entretanto, para a definição da heurística iremos considerar que após a construção das coleções, os relacionamentos reversos sempre serão criados, de forma que qualquer consulta não prevista possa ser realizada independente das coleções escolhidas, evitando desta forma o erro GCOLE.

Além disso, para a associação das consultas órfãs no final da heurística, será dada prioridade para as coleções que possuem a menor SCR, mesmo que estas sejam maiores que a de corte. Logo, estas consultas deverão utilizar relacionamento reverso para serem resolvidas.

Para formalizarmos a heurística para o OrientDB utilizando documentos ligados,

apresentada no Algoritmo 4, usaremos as mesmas definições criadas para a formalização da heurística do MongoDB.

A heurística do OrientDB utilizando documentos ligados também é muito similar à do MongoDB, onde a diferença se resume na ausência da necessidade de se incluir a coleção baseada na tabela de fatos caso alguma coleção baseada em alguma tabela de dimensão tenha sido escolhida. Portanto, na heurística do OrientDB utilizando documentos ligados, até o passo (iii) será igual à heurística do MongoDB.

No passo (iv) da heurística do OrientDB utilizando documentos ligados, as consultas que pertenciam às coleções excluídas no passo (iii), que não foram reassociadas a outras coleções, ou as consultas que ainda não haviam sido associadas a alguma coleção de R, consultas órfãs ($\{Q - Q^A\}$), serão associadas às coleções que possuem a menor SCR ou, caso esta seja igual em todas as coleções, para aquela que possui o maior número de documentos.

O passo (v) da heurística inclui a coleção baseada na tabela de fatos se não existir nenhuma coleção baseada em uma tabela de dimensão em R, ou seja, se todas juntas tinham a SCR menor que a de corte em menos de 20% das consultas e associa todas as consultas à coleção baseada na tabela de fatos.

Para exemplificar a utilização da heurística iremos aplicá-la nas coleções do SSB. Para tanto utilizaremos como valor de corte para as coleções a SCR menor ou igual a 0,2 e a tabela 4.10. O texto que será exposto no exemplo é igual ao utilizado no MongoDB, porém, como a seletividade de corte é diferente, os conjunto formados serão diferentes.

No passo (i) o conjunto R será definido como vazio.

No passo (ii) os seguintes conjuntos serão formados:

A coleção *Orderdate* possui a SCR menor que a de corte nas consultas 1.1 a 1.3 e 3.4, porém somente nas consultas 1.1 a 1.3 estas SCR são menores que as das outras coleções, logo:

- $Q_L^{Orderdate} = \{1.1, 1.2, 1.3, 3.4\}$;
- $Q^{Orderdate} = \{1.1, 1.2, 1.3\}$;
- $R = \{C^{Orderdate}\}$; e
- $Q^A = \{1.1, 1.2, 1.3\}$.

A coleção *Part* possui a SCR menor que a de corte nas consultas 2.1, 2.2, 2.3 e 4.3. Logo:

Algorithm 4 Heurística do OrientDB, utilizando documentos ligados (OL1).

1: Entrada: C^D, $\eta(C^{D_i})$, Q, $\sigma_{Q_n}(C^{D_i})$, C^F, $L^{C^{D_i}}$
2: Saida: R; Q^x,$\forall x \in C^D$; Q^{C^F}
3: **função** seleciona_modelagens(C^D, C^F, $\eta(C^{D_i})$, Q, $\sigma_{Q_n}(C^{D_i})$, $L^{C^{D_i}}$)
Passo(i):
4: $R \leftarrow \{\}$;
Passo(ii):
5: Para cada $x \in C^D$, faça:
6: Para cada $Q_k \in Q$, faça:
7: Se $\sigma_{Q_k}(x) < L^x$ então:
8: $Q_L^x \leftarrow Q_L^x \cup \{Q_k\}$;
9: existe_melhor $\leftarrow 0$
10: Para cada $y \in C^D$, onde $y \neq x$, faça:
11: Se ($\sigma_{Q_k}(y) < \sigma_{Q_k}(x)$) ou ($\sigma_{Q_k}(y) = \sigma_{Q_k}(x)$ e $\eta(y) > \eta(x)$) então:
12: existe_melhor $\leftarrow 1$
13: Pare;
14: Se !existe_melhor então:
15: $R \leftarrow R \cup \{x\}$;
16: $Q^x \leftarrow Q^x \cup \{Q_k\}$;
17: $Q^A \leftarrow Q^A \cup \{Q_k\}$;
Passo(iii):
18: $R \leftarrow$ Ordena(R);
19: Para cada $x \in R$, faça:
20: Se $|Q^x|/|Q| < 0,2$ então:
21: $R \leftarrow R - \{x\}$
22: $Q^A \leftarrow Q^A - Q^x$
23: Para cada $z \in R$, faça:
24: Para cada $Q_k \in \{R\}$, faça:
25: Se $\sigma_{Q_k}(x) < L^x$ então:
26: existe_melhor $\leftarrow 0$;
27: Para cada $y \in R$, onde $y \neq z$, faça:
28: Se ($\sigma_{Q_k}(y) < \sigma_{Q_k}(z)$) ou ($\sigma_{Q_k}(y) = \sigma_{Q_k}(z)$ e $\eta(y) > \eta(z)$) então:
29: existe_melhor $\leftarrow 1$;
30: Pare;
31: Se !existe_melhor então:
32: $Q^z \leftarrow Q^z \cup \{Q_k\}$;
33: $Q^A \leftarrow Q^A \cup \{Q_k\}$;
Passo(iv):
34: Para cada $z \in R$, faça:
35: Para cada $Q_k \in \{Q - Q^A\}$, faça:
36: existe_melhor $\leftarrow 0$;
37: Para cada $y \in R$, onde $y \neq z$, faça:
38: Se ($\sigma_{Q_k}(y) < \sigma_{Q_k}(z)$) ou ($\sigma_{Q_k}(y) = \sigma_{Q_k}(z)$ e $\eta(y) > \eta(z)$) então:
39: existe_melhor $\leftarrow 1$;
40: Pare;
41: Se !existe_melhor então:
42: $Q^z \leftarrow Q^z \cup \{Q_k\}$;
43: $Q^A \leftarrow Q^A \cup \{Q_k\}$;
Passo(v):
44: Se $R = \{\}$ então:
45: $R \leftarrow R \cup C^F$;
46: $Q^{C^F} \leftarrow Q$;

- $Q_L^{Part} = \{2.1, 2.2, 2.3, 4.3\}$.
- $Q^{Part} = \{2.1, 2.2, 2.3, 4.3\}$
- $R = \{C^{Orderdate}, C^{Part}\}$
- $Q^A = \{1.1, 1.2, 1.3, 2.1, 2.2, 2.3, 4.3\}$

Ainda no passo (ii), devido a coleção *Customer* possuir um número maior de documentos que a coleção *Supplier* e possuir nas consultas 3.1 a 4.2 a mesma SCR que esta última, também menor que a de corte, estas consultas serão adicionadas ao conjunto $Q^{Customer}$ e esta coleção também será adicionada ao conjunto R.

Logo:

- $Q_L^{Customer} = \{3.1, ..., 4.2\}$;
- $Q^{Customer} = \{3.1, ..., 4.2\}$;
- $R = \{C^{Orderdate}, C^{Part}, C^{Customer}\}$;
- $Q^A = \{1.1, ..., 4.3\}$.

No início do passo (iii) haverá a ordenação do conjunto R. Como as coleções *Orderdate* e *Part* possuem a SCR menor que a de corte em um mesmo número de consultas, mas a coleção *Orderdate* favorece um número menor de consultas essa será a primeira do conjunto R, logo:

- $R = \{C^{Orderdate}, C^{Part}, C^{Customer}\}$.

Como todas as coleções em R favorecem mais de 20% das consultas ($|Q^x|/|Q| > 0,2$), nenhuma coleção será retirada de R e não haverá necessidade de reassociar consultas de coleções excluídas.

No passo (iv), como não existem consultas órfãs, não há necessidade de associar alguma consulta a alguma coleção.

No passo (v), como existem coleções em R, a coleção C^F não será incluída.

Logo, após a aplicação da heurística existirão os seguintes conjuntos:

- $R = \{C^{Orderdate}, C^{Part}, C^{Customer}\}$;
- $Q^A = \{1.1, ..., 4.3\}$;
- $Q^{Orderdate} = \{1.1, 1.2, 1.3\}$;

- $Q^{Part} = \{2.1, 2.2, 2.3, 4.3\}$; e
- $Q^{Customer} = \{3.1, ..., 4.2\}$.

Nas tabelas 4.10 e 4.9 podemos observar que as coleções escolhidas pela a heurística permitiram que as consultas fossem realizadas pelas coleções que mais as favoreceram, e o ganho médio obtido pela utilização da heurística foi de 82%.

4.9 RESULTADOS E HEURÍSTICAS NO COUCHDB

Para analisar os resultados do CouchDB foram observados dois tempos. O tempo de geração das visões materializadas, que são necessárias para realizar as consultas, e o tempo de execução das consultas.

Na tabela 4.11 é possível observar que o tempo de geração das visões materializadas, foi aproximadamente o mesmo para todas as visões criadas em uma coleção. Notou-se que a quantidade de documentos existentes, ou o espaço de armazenamento ocupado pelas coleções, não influencia no tempo de criação da visão. A coleção *Lineorder* é a coleção que possui maior número de documentos, bem como a que ocupa maior espaço de armazenamento, mas isso não fez com que esta levasse mais tempo para construir as visões materializadas.

TAB. 4.11: Tempo de construção das visões materializadas no CouchDB em minutos.

Visões	Lineorder	Customer	Orderdate	Part	Supplier
1.1	26:09,322	29:59,034	19:52,347	17:31,035	21:48,126
1.2	26:12,239	30:18,53	19:58,88	17:35,494	21:35,353
1.3	26:10,122	30:12,778	19:48,844	17:37,116	21:31,569
2.1	26:09,753	29:51,943	20:31,576	17:30,033	21:39,605
2.2	26:09,265	30:32,914	20:21,923	17:33,686	21:34,893
2.3	25:55,531	30:31,257	20:24,528	17:35,66	21:31,543
3.1	26:08,674	29:40,813	20:29,839	17:04,812	21:50,038
3.2	26:11,052	29:54,572	20:23,919	17:35,15	21:43,474
3.3	26:02,942	30:11,787	20:21,479	17:36,127	21:32,235
3.4	26:06,392	29:58,855	19:51,647	17:38,139	21:28,653
4.1	26:11,368	30:07,054	20:36,27	17:25,069	21:41,3
4.2	26:15,193	30:4,898	20:10,441	17:34,682	21:34,316
4.3	25:57,407	30:19,526	20:04,45	17:35,284	21:31,204

Com relação aos tempos de execução das consultas sobre as visões, que podem ser visualizados na tabela 4.12, foi observado que foram próximos para uma mesma consulta em diferentes coleções, sendo influenciados principalmente pelo espaço ocupado pela visão

em disco. As visões materializadas 1.2 e 3.1 ocupam, na coleção *Lineorder*, um espaço de 82,5 KB e 1,95 MB, respectivamente. O fato da visão 1.2 ter menor tamanho fez com que o tempo de resposta da visão 1.2 fosse mais rápido. Uma vez que cada visão possui o mesmo tamanho para todas as coleções, qualquer coleção pode ser usada para gerar as visões.

TAB. 4.12: Tempo médio de execução das consultas no CouchDB em ms.

Consulta	Lineorder	Customer	Orderdate	Part	Supplier
1.1	16	15	17	15	15
1.2	11	11	10	11	11
1.3	11	10	10	10	10
2.1	84	86	82	83	78
2.2	22	25	22	22	22
2.3	9	9	9	9	9
3.1	103	93	97	116	105
3.2	100	99	105	102	82
3.3	13	13	13	13	12
3.4	9	9	9	9	9
4.1	27	23	21	22	21
4.2	41	45	35	46	42
4.3	18	18	19	17	19

Portanto, se o tempo de criação das visões é o mesmo para todas as consultas em uma coleção e o tempo de execução das consultas não varia para as diferentes coleções, não é possível estabelecer uma heurística que oriente quanto à escolha de uma modelagem.

Alternativamente, o critério de escolha poderia ser baseado no espaço de armazenamento. Então, no CouchDB, a coleção escolhida seria a coleção baseada na tabela de dimensão que possui o menor número de tuplas no modelo relacional, pois esta não só possuirá o menor número de documentos, mas a quantidade de dados em redundância também será menor quando os dados forem passados para o formato JSON.

4.10 RESULTADOS E HEURÍSTICAS NO MARKLOGIC

Em virtude do MarkLogic ser um SGBD comercial e devido a sua licença de uso não permitir a divulgação de resultados de performance ou estatísticos deste SGBD, os resultados do comportamento deste em diferentes modelagens somente serão discutidos a partir da tabela de SCR das coleções e das coleções que obtiveram o melhor desempenho nas consultas.

Analisando a tabela 4.13 é possível observar que os resultados de poucas consultas

tiveram alguma relação com os atributos utilizados ou com as seletividades dos seus filtros. Por isso, não foi possível encontrar um padrão no comportamento do MarkLogic quando variando a modelagem do agregado. Um exemplo de comportamento contraditório é o das consultas 3.1 e 3.2. Na cláusula de seleção da consulta 3.1 há um campo raiz da coleção *Customer*, com seletividade 0,2. Esta consulta obteve o melhor tempo nesta coleção. Já a consulta 3.2, que possui um campo raiz da mesma coleção *Customer*, com seletividade 0,04, teve melhor desempenho sobre a coleção *Lineorder*.

Outro exemplo de comportamento contraditório é observado nas consultas 2.2 e 2.3. Na consulta 2.2, o filtro utilizado no atributo *s_region* possui seletividade 0,2 e o tempo de execução desta consulta na coleção *Supplier* foi menor que na coleção *Lineorder*. Já na consulta 4.3, embora o filtro utilizado no atributo *p_category* possua seletividade 0,008, a coleção *Lineorder* obteve o menor tempo nesta consulta.

TAB. 4.13: Seletividade dos campos raízes das coleções do SSB, e no final, o número de documentos em cada coleção de agregado.

Consultas	Lineorder	Customer	Orderdate	Part	Supplier
1.1	0,166		0,142		
1.2	0,045		0,011		
1.3	0,045		0,002		
2.1				0,04	0,2
2.2				0,008	0,2
2.3				0,001	0,2
3.1		0,2	0,875		0,2
3.2		0,04	0,875		0,04
3.3		0,008	0,875		0,008
3.4		0,008	0,011		0,008
4.1		0,2		0,4	0,2
4.2		0,2	0,25	0,4	0,2
4.3		0,2	0,25	0,008	0,04
Documentos	6001215	30000	2556	200000	2000

Portanto, devido à não observação de um comportamento padrão por parte do SGBD MarkLogic, não foi possível estabelecer uma heurística de modelagem de agregados para otimizar consultas analíticas neste SGBD.

4.11 DISCUSSÕES A RESPEITO DOS EXPERIMENTOS INICIAIS

A partir do experimentos nos cinco SGBD foi possível observar que o impacto da variação da modelagem foi influenciado por alguns aspectos similares em alguns SGBD e por aspectos diferentes em outros.

Nos SGBD MongoDB e OrientDB, tanto utilizando documentos aninhados, quanto documentos ligados, o principal fator que influenciou no comportamento dos SGBD foi a SCR dos filtros utilizados nas consultas. E devido a esse aspecto em comum, as heurísticas criadas para esses SGBD foram muito similares.

Nas heurísticas propostas para esses SGBD são utilizados valores de corte para as coleções baseadas nas tabelas de dimensão, na qual para as consultas que possuem filtros cuja SCR das coleções baseadas nas tabelas de dimensão são menores que este valor de corte, essas coleções possuirão melhor desempenho do que a coleção baseada na tabela de fatos.

Porém, este valor não pode ser generalizado para todos os conjuntos de dados, e devido à impossibilidade deste ser obtido matematicamente, pode-se alternativamente, após a criação das primeiras coleções, baseadas em uma seletividade de corte inicial estimada, obter este valor experimentalmente e reaplicar a heurística de forma a escolher novas modelagens de agregado e melhorar ainda mais o desempenho das consultas.

À medida que os dados são alterados nas diversas coleções é provável que a seletividade de corte também se altere. Logo, é necessário que o valor da seletividade de corte das coleções seja recalculada de tempos em tempos, de forma que as heurísticas sejam melhor aplicadas.

No ElasticSearch também foi encontrado um padrão no seu comportamento, porém a heurística proposta para ele baseou-se em aspectos diferentes do que os SGBD citados anteriormente.

Apesar de ter sido encontrado um comportamento padrão no CouchDB, não foi possível propor uma heurística, principalmente pelo fato deste SGBD utilizar visões materializadas para a realização das consultas.

Dentre os SGBD estudados, somente no MarkLogic não foi encontrado um padrão no seu comportamento e por isso também não foi possível construir uma heurística para este SGBD.

As heurísticas propostas tinham como objetivo orientar os usuários na escolha de um mínimo de modelagens que pudessem incrementar somente o desempenho das consultas nos SGBD estudados. Entretanto, sabe-se que a manutenção de muitas coleções em diferentes modelagens poderá causar um impacto em outras operações realizadas sobre os SGBD. Apesar de não ter sido mensurado neste trabalho, foi considerada a existência deste custo, uma vez que o trabalho busca evitar que todas as coleções com as possíveis modelagens sejam mantidas. Uma evidência disso foi o uso do princípio de Pareto como um parâmetro para eliminar as coleções que favorecem menos de 20% das consultas e

assim diminuir o custo de manutenção das coleções restantes.

Além disso, as heurísticas propostas foram construídas a partir do comportamento dos SGBD sobre um conjunto de dados que cabia na memória principal da máquina utilizada, portanto não se pode afirmar que as heurísticas também serão válidas para conjuntos de dados maiores.

5 VALIDAÇÃO DE HEURÍSTICAS

Conforme mencionado no capítulo 2, para a validação das heurísticas propostas ou confirmação do comportamento de um SGBD, foi escolhido o conjunto de dados e algumas consultas do *benchmark* TPC-DS. A sua escolha se deve ao fato de possuir um objetivo similar ao do SSB (avaliar aplicações analíticas). Entretanto, devido o TPC-DS possuir sete esquemas de dados, mais complexos que o esquema do SSB, com algumas consultas que envolvem mais de um esquema, algumas alterações no conjunto de dados do TPC-DS foram necessárias.

5.1 ALTERAÇÕES NO TPC-DS PARA VALIDAÇÃO DAS CONSULTAS

O TPC-DS possui no total sete esquemas de dados, onde as consultas por vezes utilizam um ou mais esquemas que podem ser relacionados, porém foi escolhido somente um esquema para a realização dos experimentos e selecionadas algumas consultas que utilizavam as tabelas deste esquema. Além disso, tendo em vista a necessidade de limitar o trabalho no estudo de agregados com dados em profundidade de até três níveis, foi necessário realizar a desnormalização de algumas tabelas do esquema escolhido.

O esquema de dados escolhido foi o esquema de venda de lojas que pode ser observado na figura 5.1, chamado de *Store Sales* pelo TPC-DS. Foram realizadas as junções das tabelas *Promotion*, *Store* e *Store_Sales* em uma nova tabela *Store_Sales*. As junções das tabelas *Customer*, *Customer_Demographics*, *Customer_Address* e *Household_Demographics*, originaram uma nova tabela *Customer*. O novo esquema de dados é apresentado na Figura 5.2. Além dessas modificações, foram eliminadas da tabela *Store_Sales* gerada, as tuplas que continham dados incompletos, como as tuplas cujas chaves estrangeiras continham nulos. Dessa forma, garantimos a uniformidade das respostas em todos as modelagens.

O TPC-DS possui um total de 99 consultas analíticas que utilizam os sete esquemas existentes. Para validação das heurísticas foram selecionadas somente aquelas consultas sobre o esquema de dados escolhido (Store_Sales) e algumas consultas que utilizam mais de um esquema, dentre eles o Store_Sales. Porém, para estas últimas foi necessário realizar algumas alterações de forma que fossem eliminadas as sub-consultas sobre os outros esquemas.

Um exemplo de modificação de consulta pode ser observado na consulta 71 do TPC-

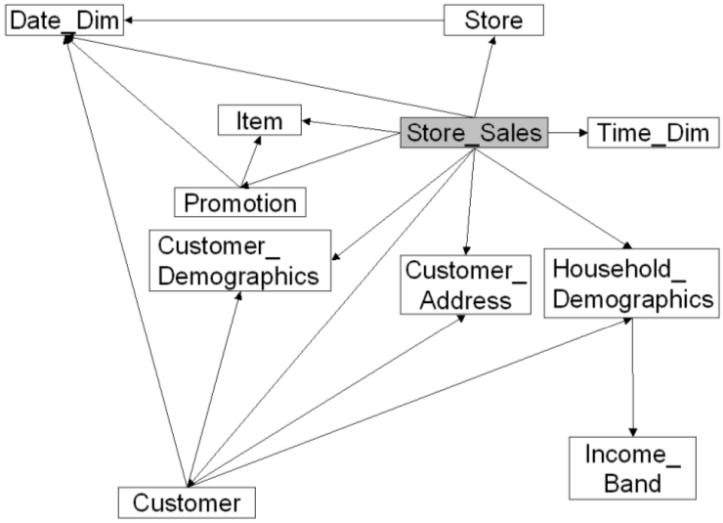

FIG. 5.1: Esquema Store Sales original do TPC-DS.

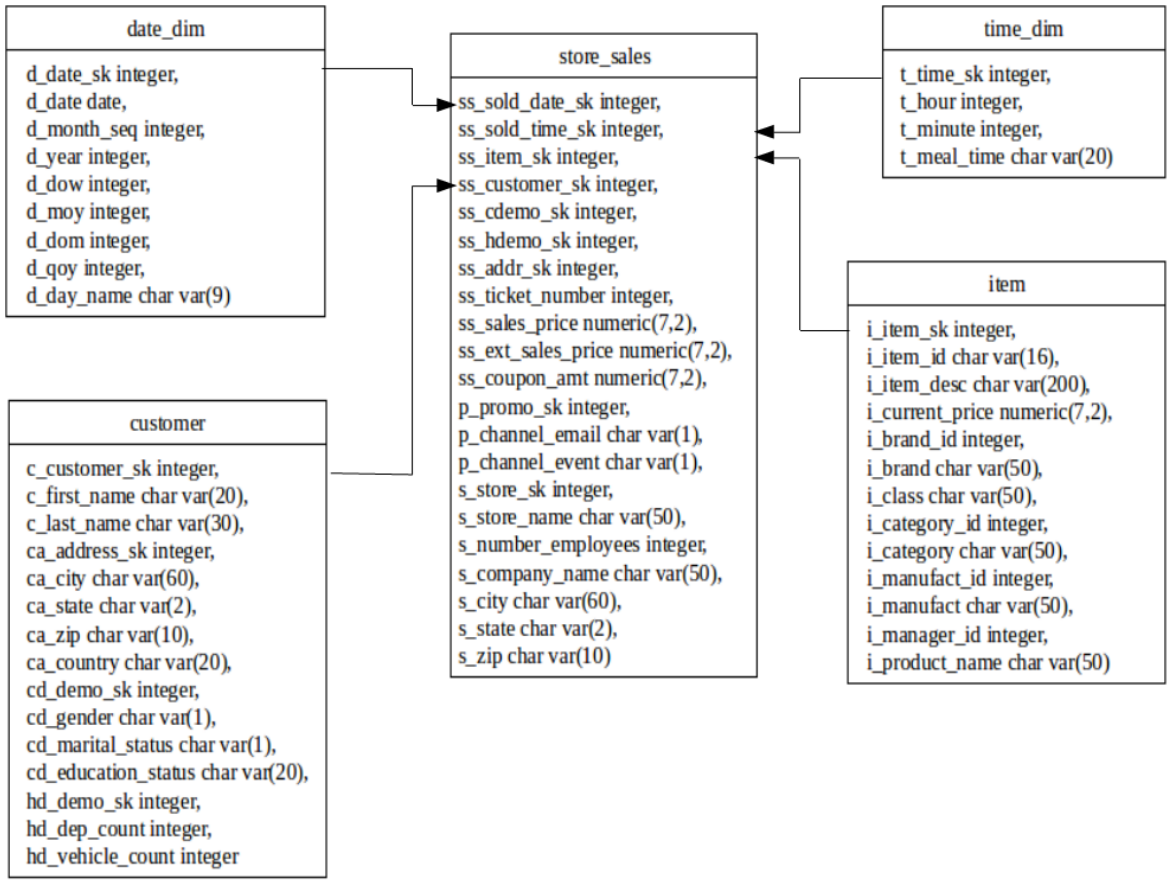

FIG. 5.2: Esquema Store Sales alterado.

DS abaixo apresentado. Na sua forma original havia uma união de 3 sub-consultas em 3 diferentes esquemas. Na consulta modificada, somente a sub-consulta onde a cláusula FROM envolvia a tabela Store_Sales (em negrito) foi mantida e as outras sub-consultas

foram eliminadas. Em outras consultas, devido à desnormalização de algumas tabelas, algumas junções entre tabelas que deixaram de existir também foram eliminadas. Além disso, foram estabelecidos valores arbitrários para as variáveis das consultas do TPC-DS, como por exemplo nas variáveis MONTH e YEAR da consulta 71, seguindo os valores estabelecidos na documentação do TPC-DS.

Query 71 Original:

```
SELECT i_brand_id brand_id, i_brand brand,
t_hour,t_minute, sum(ext_price) ext_price
FROM item,
(SELECT ws_ext_sales_price as ext_price,
ws_sold_date_sk as sold_date_sk,
ws_item_sk as sold_item_sk,
ws_sold_time_sk as time_sk
FROM web_sales, date_dim
WHERE d_date_sk =ws_sold_date_sk AND
d_moy=[MONTH] AND d_year=[YEAR]
UNION ALL
SELECT cs_ext_sales_price as ext_price,
cs_sold_date_sk as sold_date_sk,
cs_item_sk as sold_item_sk,
cs_sold_time_sk as time_sk
FROM catalog_sales, date_dim
WHERE d_date_sk=cs_sold_date_sk AND
d_moy=[MONTH] AND d_year=[YEAR]
UNION ALL
SELECT
ss_ext_sales_price as ext_price,
ss_sold_date_sk as sold_date_sk,
ss_item_sk as sold_item_sk,
ss_sold_time_sk as time_sk,
FROM store_sales, date_dim
WHERE d_date_sk = ss_sold_date_sk AND
d_moy=[MONTH] AND
d_year=[YEAR]) as tmp,time_dim
WHERE sold_item_sk=i_item_sk AND
i_manager_id=1 AND time_sk = t_time_sk AND
(t_meal_time='breakfast'
OR t_meal_time='dinner')
GROUP BY i_brand, i_brand_id,t_hour,t_minute
ORDER BY ext_price desc, i_brand_id ;
```

Query 71 Adaptada:

```
SELECT i_brand_id brand_id,i_brand brand,
t_hour,t_minute,
sum(ss_ext_sales_price) ext_price
FROM item,date_dim,store_sales,time_dim
WHERE ss_item_sk = i_item_sk AND
ss_sold_time_sk=t_time_sk AND
ss_sold_date_sk= d_date_sk AND
i_manager_id=1 AND d_moy=11 AND
```

```
d_year=2001 AND (t_meal_time='breakfast' OR
t_meal_time='dinner')
GROUP BY i_brand,i_brand_id,t_hour,t_minute;
```

As consultas do TPC-DS selecionadas para este trabalho foram as consultas 3, 6, 19, 27, 53, 68, 71, 79, 88, 89, 96 e 98 onde as suas formas originais e as formas finais, após as adaptações, podem ser consultadas no apêndice 2.

5.2 CRIAÇÃO DOS AGREGADOS DOS DADOS DO TPC-DS EM DIFERENTES MODELAGENS

A partir do novo esquema de dados do TPC-DS é possível criar 5 coleções com modelagens diferentes de agregados, da mesma forma que no SSB. Cada coleção é baseada em uma tabela de dimensão, havendo portanto cinco coleções: *Store_Sales*, *Customer*, *Item*, *Date* e *Time*.

A escala de geração de dados utilizada para o TPC-DS também foi a escala 1, gerando 2,880,404 tuplas para a tabela *Store_Sales*, 100,000 tuplas para a tabela *Customer*, 73,049 tuplas para a tabela *Date_dim*, 18,000 tuplas para a tabela *Item* e 86,400 tuplas para a tabela *Time_dim*. Da mesma forma que no SSB, o espaço ocupado pelos dados no formato JSON, quanto nos diferentes SGBD foi um pouco maior. Apesar de ter sido gerado 1 GB de dados na escala 1 do gerador, somente 560 MB foram realmente aproveitados, em virtude de algumas tabelas não serem utilizadas pelo esquema *Store_Sales*. A tabela 5.1 apresenta o espaço ocupado pelos documentos no formato JSON e o espaço ocupado em cada SGBD.

TAB. 5.1: Tamanho das coleções de agregados em Gigabytes no formato JSON e dentro de cada SGBD

	Customer	Date	Item	Time	Store_Sales
JSON	2,9	3,4	2,7	3,6	3,8
MongoDB	0,903	0,874	0,678	0,887	1,081
ElasticSearch	1,9	2,1	1,8	2,1	2,2
OrientDB documento	3,7	3,7	3,1	4,2	5,6
OrientDB ligado	6,2	5,6	4,6	5,8	6,7
CouchDB	0,8	1,1	0,7	0,9	3,3
MarkLogic	4,4	3,9	3,4	4,4	9,3

Da mesma forma que no SSB, cada modelagem de agregado será intitulada com o nome da tabela que foi utilizada de base para a contrução da modelagem, bem como a coleção que possui documentos modelados segundo uma modelagem de agregado.

5.3 AJUSTE DE CONSULTAS DO TPC-DS

Devido às consultas a serem utilizadas para validação das heurísticas possuírem um sintaxe equivalente às consultas do SSB, não serão demonstradas como foram feitas as adaptações para as consultas do TPC-DS para cada SGBD. Porém cabe ressaltar que devido à coleção *Store_Sales* ser a coleção baseada na tabela de fatos no TPC-DS, a operação *unwind*, tanto no MongoDB como no OrientDB e a operação *inner hits* no ElasticSearch não são necessárias nesta coleção.

As consultas nos SGBD aqui estudados, nas diferentes modelagens de agregados podem ser consultadas no seguinte endereço eletrônico: https://github.com/raphha/Doc_Store_TPC_DS.

5.4 CONFIGURAÇÕES PARA OS EXPERIMENTOS DE VALIDAÇÃO

Para a validação dos dados foi utilizada a mesma máquina dos experimentos iniciais e as configurações dos diferentes SGBD foram mantidas.

Os resultados apresentados por ocasião da validação serão apresentados da mesma forma que nos experimentos iniciais. Sendo relevante a observação que a validação das heurísticas se dará somente para as consultas realizadas sequencialmente, sendo excluídos os tempos das primeiras consultas, pois as heurísticas foram construídas com base nos resultados de consultas realizadas sequencialmente.

5.5 ENTENDIMENTO DOS RESULTADOS DE VALIDAÇÃO APRESENTADOS

No capítulo anterior, foi possível estabelecer heurísticas de modelagem de agregados para escolher as coleções com as quais se pode obter uma melhor performance em consultas analíticas em alguns SGBD.

Nas tabelas de resultados neste capítulo, diferente dos resultados apresentados nos experimentos iniciais, os tempos em amarelo representam os tempos das coleções escolhidas para as consultas segundo a heurística proposta e os tempos em verde, quando existirem, representam a coleção que obteve o melhor tempo naquela consulta, que não seguiu a heurística proposta. Os ganhos apresentados serão os obtidos ao utilizar a heurística proposta comparados com os tempos das consultas realizadas na modelagem baseada na tabela de fatos.

5.6 VALIDAÇÃO DA HEURÍSTICA DO MONGODB

Para validação da Heurística do MongoDB, devido aos dados do TPC-DS serem similares aos dados do SSB, o valor da seletividade de corte utilizado inicialmente para as coleções foi o valor de 0,1, e a partir dos valores da SCR das coleções, que podem ser visualizadas na tabela 5.2, os passos da heurística criada para o MongoDB foram seguidos.

TAB. 5.2: Seletividade dos campos raízes das coleções do TPC-DS, e no final, o número de documentos em cada coleção de agregado.

Consultas	Store_Sales	Customer	Date	Item	Time
3			0,2	0,0008	
6			0,2	0,16	
19		0,0009	0,032	0,019	
27	0,99	0,013	0,2		
53			0,25	0,303	
68	0,833	0,25	0,04		
71			0,032	0,017	0,5
79		0,387	0,084		
88		0,27			0,019
89			0,201	0,112	
96		0,09			0,041
98			0,0121	0,303	
Documents	2652870	100000	73049	18000	86400

A partir desses valores iremos seguir os passos da heurística M1 visando selecionar as melhores modelagens de agregado para as consultas do TPC-DS:

Passo (i):

Neste passo o conjunto R será definido como um conjunto vazio.

Passo (ii):

Os seguintes conjuntos serão formados se considerarmos que conjunto C^D está ordenado segundo a tabela 5.2, onde o primeiro elemento é a coleção *Customer*:

Coleção *Customer*:

- $Q_L^{Customer} = \{19, 27, 96\}$;
- $Q^{Customer} = \{19, 27\}$;
- $Q^A = \{19, 27\}$;
- $R = \{C^{Customer}\}$;

Coleção *Date*:

- $Q_L^{Date} = \{19, 68, 71, 79, 98\}$;
- $Q^{Date} = \{68, 79, 98\}$;
- $Q^A = \{19, 27, 68, 79, 98\}$;
- $R = \{C^{Customer}, C^{Date}\}$;

Coleção *Item*:

- $Q_L^{Item} = \{3, 19, 71\}$;
- $Q^{Item} = \{3, 71\}$;
- $Q^A = \{19, 27, 68, 79, 98, 3, 71\}$;
- $R = \{C^{Customer}, C^{Date}, C^{Item}\}$;

Coleção *Time*:

- $Q_L^{Time} = \{88, 96\}$;
- $Q^{Time} = \{88, 96\}$;
- $Q^A = \{19, 27, 68, 79, 98, 3, 71, 88, 96\}$;
- $R = \{C^{Customer}, C^{Date}, C^{Item}, C^{Time}\}$;

Passo (iii):

Inicialmente o conjunto R será ordenado da seguinte forma:

- $R = \{C^{Time}, C^{Item}, C^{Customer}, C^{Date}\}$;

Como $|Q^{Time}|/|Q| < 0,2$ a coleção *Time* será descartada de R. Como, dentre as consultas de Q^{Time}, a coleção *Customer* possui a SCR menor que a de corte na consulta 96, esta consulta será adicionada a $Q^{Customer}$ e a consulta 88 será excluída de Q^A. Logo $Q^{Customer} = \{19, 27, 96\}$ e $Q^A = \{19, 27, 68, 79, 98, 3, 71, 96\}$;

A próxima coleção em R é *Item* e como $|Q^{Item}|/|Q| < 0,2$, esta também será descartada. Como a consulta 71 possui a SCR menor que a de corte na coleção *Date*, esta passará a executar esta consulta e a consulta 3 será excluída de Q^A. Logo $Q^{Date} = \{68, 79, 98, 71\}$ e $Q^A = \{19, 27, 68, 79, 98, 71, 96\}$;

Como $|Q^x|/|Q| > 0,2$ para as outras coleções $x \in R$, nenhuma outra coleção será excluída de R.

Passo (iv): Como $|Q^A|/|Q| < 0,8$ a coleção C^{Store_Sales} será incluída em R.

- $R = \{C^{Time}, C^{Item}, C^{Customer}, C^{Store\_Sales}\}$

Passo (v): Como $C^{store\_Sales} \in R$, então $Q^{Store\_Sales} = Q - Q^A = \{3, 6, 53, 88, 89\}$.

Ao final, após seguir todos os passos da Heurística M1, as coleções *Store_Sales*, *Customer* e *Date* serão escolhidas para melhorar o desempenho das consultas, onde cada uma deverá executar as seguintes consultas:

- $Q^{Store\_Sales} = \{3, 6, 53, 88, 89\}$;
- $Q^{Date} = \{68, 79, 98, 71\}$;
- $Q^{Customer} = \{19, 27, 96\}$

A partir da tabela 5.3 é possível observar que apesar de as modelagens escolhidas a partir das heurísticas não serem as melhores para todas as consultas do TPC-DS, o ganho médio obtido ao utilizar a heurística foi de 42%.

Ao aplicar a heurística, as consultas 3, 71 e 89 não seriam associadas pela coleção *Item* que mais as favoreceriam. Ao invés, a coleção escolhida para executá-las seria a coleção *Store_Sales* que é a coleção que obteve o segundo melhor desempenho nessas consultas, permitindo ainda um bom desempenho para elas.

Apesar da maioria das consultas órfãs terem obtido bons desempenhos ao serem executadas pela coleção baseada na tabela de fato, algumas exceções poderão ocorrer. A consulta 88 é um exemplo deste comportamento. Esta consulta sobre a coleção *Customer* obteve um desempenho melhor que sobre a coleção *Store_Sales*, mesmo com uma SCR bem acima da seletividade de corte.

Na consulta 88 há uma combinação de filtros, cujos campos são raízes de diferentes coleções: *Time* e *Customer*. Na coleção *Customer*, a quantidade de campos aninhados nos quais foi necessário realizar a operação *unwind* foi pequena, sendo 2% menor do que na coleção *Time*, sendo este o motivo pelo qual esta consulta obteve um bom desempenho na coleção *Customer*.

Na tabela 5.4 podemos observar os valores da seletividade de corte encontrados para as coleções formadas a partir do esquema do TPC-DS. Nesta tabela podemos notar que a seletividade de corte ficou próxima de 0,1, o que comprova que para os dados do TPC-DS, esse valor de corte foi uma boa estimativa inicial.

Assim, pode-se dizer que com base no experimento realizado com o TPC-DS foi possível validar a heurística proposta para o MongoDB.

TAB. 5.3: Média do tempo de execução das consultas do TPC-DS no MongoDB.

Consultas	Store_Sales	Customer	Date	Item	Time	Ganho
3	3654	3688	10202	1598	5134	0%
6	4413	16575	11089	5263	26308	0%
19	3151	220	526	275	2973	93%
27	3909	535	8148	19771	16038	86%
53	6951	27469	11523	9160	34745	0%
68	6694	2639	2305	32039	16144	66%
71	3707	6541	1402	588	8477	62%
79	7555	6314	4842	29387	26335	36%
88	8415	1388	45962	36387	839	0%
89	8581	20044	9774	3509	28513	0%
96	3502	738	40082	29298	1454	79%
98	5103	16011	1001	6915	26401	80%
Média	5470	8514	12238	14516	16564	42%

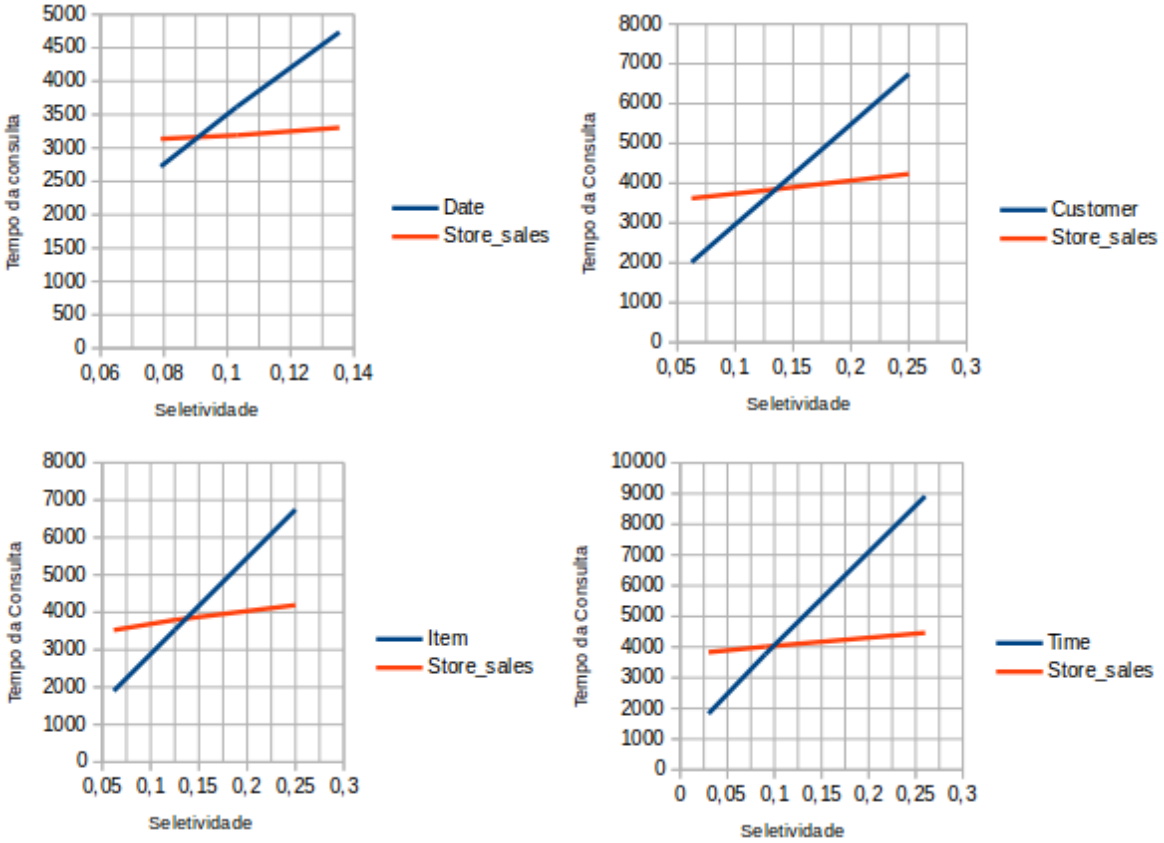

FIG. 5.3: Gráficos Tempo x Seletividade no MongoDB.

5.7 VALIDAÇÃO DA HEURÍSTICA DO ELASTICSEARCH

Na tabela 5.5, podemos observar as coleções que possuem campos raízes nas cláusulas de seleção das consultas do TPC-DS e a partir desta tabela podemos seguir os passos da

TAB. 5.4: Seletividade de Corte das coleções do TPC.

Coleção	Número de documentos	Seletividade de Corte
Customer	100000	0,13
Time	86400	0,1
Date	73049	0,09
Item	1800	0,13

heurística E1 e escolher algumas modelagens de agregado que favorecem boa parte das consultas do TPC-DS.

TAB. 5.5: Seletividade dos campos raízes das coleções do TPC-DS.

Consultas	Store_Sales	Customer	Date	Item	Time
3			X	X	
6			X	X	
19		X	X	X	
27	X	X	X		
53			X	X	
68	X	X	X		
71			X	X	X
79		X	X		
88		X			X
89			X	X	
96		X			X
98			X	X	

No passo (i) da heurística seria atribuído vazio aos conjuntos R e Q^A.

No passo (ii) como todas as consultas do TPC-DS possuíam pelo menos um campo raiz das tabelas baseadas nas tabelas de dimensão, nas suas cláusulas de seleção, os conjuntos R, Q^A e Q^{C^F} permaneceram vazios e $Q_D = Q$.

No final do passo (iii) os seguintes conjuntos foram formados:

A coleção *Date* possui campos raízes nas consultas 3, 6, 19, 27, 53, 68, 71, 79, 89 e 98 e como esta coleção possui campos raízes na cláusula de seleção do maior número de consultas ($|Q_R^{Date}| = 10$) esta deve ser adicionada a R, logo;

- $R = \{C^{Date}\}$
- $Q^{Date} = \{3, 6, 19, 27, 53, 68, 71, 79, 89, 98\}$
- $Q^A = \{3, 6, 19, 27, 53, 68, 71, 79, 89, 98\}$

Ainda no passo (iii) como Q^A ainda é diferente de Q_D. Deve-se executar o laço novamente para as consultas 88 e 96. Como a coleção *Customer* e *Time* possuem campos

raízes nas cláusulas de seleção das duas consultas, qualquer uma poderia ser adicionada ao conjunto R. Portanto, supondo que coleção *Customer* fosse a primeira do conjunto R, esta seria a coleção que seria incluída no conjunto R. Logo:

- $R = \{C^{Date}, C^{Customer}\}$
- $Q^{Customer} = \{88, 96\}$
- $Q^A = \{3, 6, 19, 27, 53, 68, 71, 79, 89, 98, 88, 96\}$

Como neste ponto $Q_D \subset Q^A$ passaremos para o passo (iv).

No passo (iv), como $|Q^{Customer}|/|Q| < 0,2$, ou seja, a coleção *Customer* possui associada a esta menos de 20% das consultas, é necessário descartar esta coleção e as consultas 88 e 96 associadas a esta serão retiradas de Q^A e passarão a ser consultas órfãs.

No passo (v), as consultas 88 e 96 serão associadas à coleção *Date*, pois esta é a que possui o maior número de consultas associadas.

Portanto, ao final da aplicação da heurística teremos os seguintes conjuntos:

- $R = \{C^{Date}\}$
- $Q^{Date} = \{3, 6, 19, 27, 53, 68, 71, 79, 89, 98, 88, 96\}$
- $Q^A = \{3, 6, 19, 27, 53, 68, 71, 79, 89, 98, 88, 96\}$

Observando a tabela 5.6 podemos observar que o ganho obtido pela utilização da heurística na escolha das modelagens de agregados foi de 48%, mesmo tendo sido escolhida somente uma coleção para aumentar o desempenho das consultas.

Apesar de parte das consultas associadas à coleção *Date* terem obtido melhores tempos em outras coleções, a heurística proposta obteve bons resultados. Pois como observado na seção 4.7, não é possível prever qual coleção será mais rápida em uma consulta, mas somente se terá um desempenho melhor que a coleção baseada na tabela de fatos.

Além disso, outro aspecto que pode ser notado na tabela 5.6 é o fato de que a média do tempo de execução das consultas na coleção *Item* ter sido menor do que na coleção *Date*, escolhida pela heurística. Entretanto, este fato somente ocorreu devido à consulta 53 ter obtido um tempo muito mais baixo na coleção *Item*. Se eliminarmos os tempos desta consulta da média, a coleção *Date* seria 25% mais rápida que a coleção *Item*.

Desta forma, pode-se dizer que com base no experimento realizado com o TPC-DS foi possível validar a heurística proposta para o ElasticSearch.

TAB. 5.6: Média dos tempos de execução das consultas do TPC-DS no ElasticSearch em milissegundos.

Consulta	Store_Sales	Customer	Date	Item	Time	Ganho
3	50	22	18	13	16	64%
6	60	121	22	25	38	63%
19	46	15	14	13	15	70%
27	59	25	28	145	27	52%
53	1104	1340	1104	255	1321	0%
68	275	132	80	568	87	71%
71	56	22	19	22	33	66%
79	294	114	110	289	118	63%
88	58	26	32	38	16	45%
89	78	62	66	44	67	15%
96	53	16	18	20	14	66%
98	653	628	625	214	618	4%
Média	232	210	178	137	198	48%

5.8 VALIDAÇÃO DAS HEURÍSTICAS DO ORIENTDB

5.8.1 VALIDAÇÃO DA HEURÍSTICA DO ORIENTDB UTILIZANDO DOCUMENTOS ANINHADOS

Para validação da heurística do OrientDB utilizando documentos aninhados os passos da heurística serão seguidos para o conjunto de coleções obtidos a partir do esquema do TPC-DS, e devido à impossibilidade de se calcular a seletividade de corte das coleções antes da construção dos agregados nas diferentes modelagens, será utilizado o valor de 0,05 como o valor de corte inicial. Para valores acima deste, a modelagem escolhida deve ser a modelagem baseada na tabela de fatos.

Portanto, a partir da tabela 5.7 iremos seguir os passos da heurística M1 visando selecionar as melhores modelagens de agregado para as consultas do TPC-DS:

Passo (i):

Neste passo o conjunto R será inicializado com a coleção baseada na tabela de fatos.

- $R = \{C^{STore\_Sales}\}$;

Passo (ii):

Os seguintes conjuntos serão formados quando considerarmos a ordenação do conjunto C^D segundo a tabela 5.2, onde o primeiro elemento é a coleção *Customer*:

Coleção *Customer*:

- $Q_L^{Customer} = \{19, 27\}$;

TAB. 5.7: Seletividade dos campos raízes das coleções do TPC-DS, e no final, o número de documentos em cada coleção de agregado.

Consultas	Store_Sales	Customer	Date	Item	Time
3			0,2	0,0008	
6			0,2	0,16	
19		0,0009	0,032	0,019	
27	0,99	0,013	0,2		
53			0,25	0,303	
68	0,833	0,25	0,04		
71			0,032	0,017	0,5
79		0,387	0,084		
88		0,27			0,019
89			0,201	0,112	
96		0,09			0,041
98			0,0121	0,303	
Documents	2652870	100000	73049	18000	86400

- $Q^{Customer} = \{19, 27\}$;
- $Q^{A} = \{19, 27\}$;
- $R = \{C^{Customer}, C^{STore\_Sales}\}$;

Coleção *Date*:

- $Q_L^{Date} = \{19, 68, 71, 98\}$;
- $Q^{Date} = \{68, 98\}$;
- $Q^{A} = \{19, 27, 68, 98\}$;
- $R = \{C^{Customer}, C^{Date}, C^{STore\_Sales}\}$;

Coleção *Item*:

- $Q_L^{Item} = \{3, 19, 71\}$;
- $Q^{Item} = \{3, 71\}$;
- $Q^{A} = \{19, 27, 68, 98, 3, 71\}$;
- $R = \{C^{Customer}, C^{Date}, C^{Item}, C^{STore\_Sales}\}$;

Coleção *Time*:

- $Q_L^{Time} = \{88, 96\}$;
- $Q^{Time} = \{88, 96\}$;
- $Q^A = \{19, 27, 68, 98, 3, 71, 88, 96\}$;
- $R = \{C^{Customer}, C^{Date}, C^{Item}, C^{Time}, C^{STore\_Sales}\}$;

Passo (iii):

No início desse passo o conjunto R será ordenado da seguinte forma:

- $R = \{C^{Customer}, C^{Time}, C^{Item}, C^{Date}, C^{STore\_Sales}\}$;

Como $|Q^{Customer}|/|Q| < 0,2$, a coleção *Customer* será descartada de R. Como dentre as consultas de $Q^{Customer}$ a coleção *Item* possui a menor SCR, que também é menor que a de corte na consulta 19, esta consulta será adicionada a Q^{Item} e a consulta 27 será excluída de Q^A, logo:

- $Q^{Item} = \{3, 71, 19\}$;
- $Q^A = \{19, 68, 98, 3, 71, 88, 96\}$;
- $R = \{C^{Time}, C^{Item}, C^{Date}, C^{STore\_Sales}\}$;

A próxima coleção em R é a coleção *Time*. Como $|Q^{Time}|/|Q| < 0,2$, esta coleção também será descartada. Como todas as outras coleções existentes em R possuem a SCR maior que 0,05 nas consultas 88 e 96, estas consultas não serão associadas a outras coleções baseadas nas tabelas de dimensão. Assim, temos que:

- $Q^A = \{19, 68, 98, 3, 71\}$;
- $R = \{C^{Item}, C^{Date}, C^{STore\_Sales}\}$;

Após a exclusão da coleção *Time*, verifica-se que a coleção *Item* possui $|Q^{Item}|/|Q| > 0,2$, porém a coleção *Date* não, logo a coleção *Date* também será retirada de R.

Ao excluir *Date*, a única coleção restante em R é a coleção *Item*. Como esta coleção possui a SCR maior que a de corte para as consultas 68 e 98 retiradas de Q^A (pois pertenciam a Q^{Date}), estas consultas permanecerão órfãs. Neste ponto temos então que:

- $Q^A = \{19, 3, 71\}$;
- $R = \{C^{Item}, C^{STore\_Sales}\}$;

Após a exclusão de *Date*, como $|Q^{Item}|/|Q| > 0,2$ a coleção *Item* permanecerá em R.

Passo (v):

Nesse passo, a consultas órfãs serão associadas à coleção *Store_Sales*, logo:

$Q^{Store\_Sales} = Q - Q^A = \{6, 27, 53, 68, 79, 88, 89, 96, 98\}$.

Ao final, após seguir todos os passos da Heurística OA1, as coleções *Store_Sales*, *Item* serão escolhidas a partir da heurística e as seguintes consultas estarão associadas às coleções:

- $Q^{Store\_Sales} = \{6, 27, 53, 68, 79, 88, 89, 96, 98\}$.
- $Q^{Item} = \{3, 19, 71\}$
- $R = \{C^{Item}, C^{STore\_Sales}\}$;

TAB. 5.8: Média dos tempos de execução das consultas do TPC-DS no OrientDB em milissegundos.

Consulta	Store_Sales	Customer	Date	Item	Time	Ganho
3	61465	86275	116757	25979	111247	57%
6	73699			45219		0%
19	74186	28262	32457	26174	40253	64%
27	78887	28938	517967		50258	0%
53	61465					0%
68	80450	32962	35814			0%
71	68697	37864	38903	27873	40147	59%
79	83066	43391	47384			0%
88	68581	31247			32543	0%
89	64584			24228		0%
96	61465	29627			32706	0%
98	83066	121873	33131	102562		0%
Média	74695	48937	117487	42005	51192	15%

Comparando-se as modelagens selecionadas pela heurística OA1 na tabela 5.7 e os resultados das consultas dos TPC-DS nas diferentes coleções na tabela 5.8, observa-se que o ganho obtido pela utilização da heurística foi de apenas 15%, sendo portanto um valor muito baixo.

O principal motivo para a heurística não ter conseguido um bom aumento no desempenho das consultas, ocorreu em virtude da seletividade de corte utilizada. Usou-se o valor obtido no experimento com o SSB. Após recalculá-lo para todas as coleções do TPC-DS, verificou-se que o valor inicial estava bem abaixo do valor de corte real dessas coleções.

Nos gráficos da figura 5.4, podemos observar que o menor valor de corte encontrado para as coleções foi de 0,125 aproximadamente. Com base neste valor, coleções antes descartadas, seriam mantidas. Logo, muitas consultas passariam a ser associadas às coleções que melhor as favoreceriam. Assim, após ajustar o valor da seletividade de corte, maiores ganhos seriam obtidos.

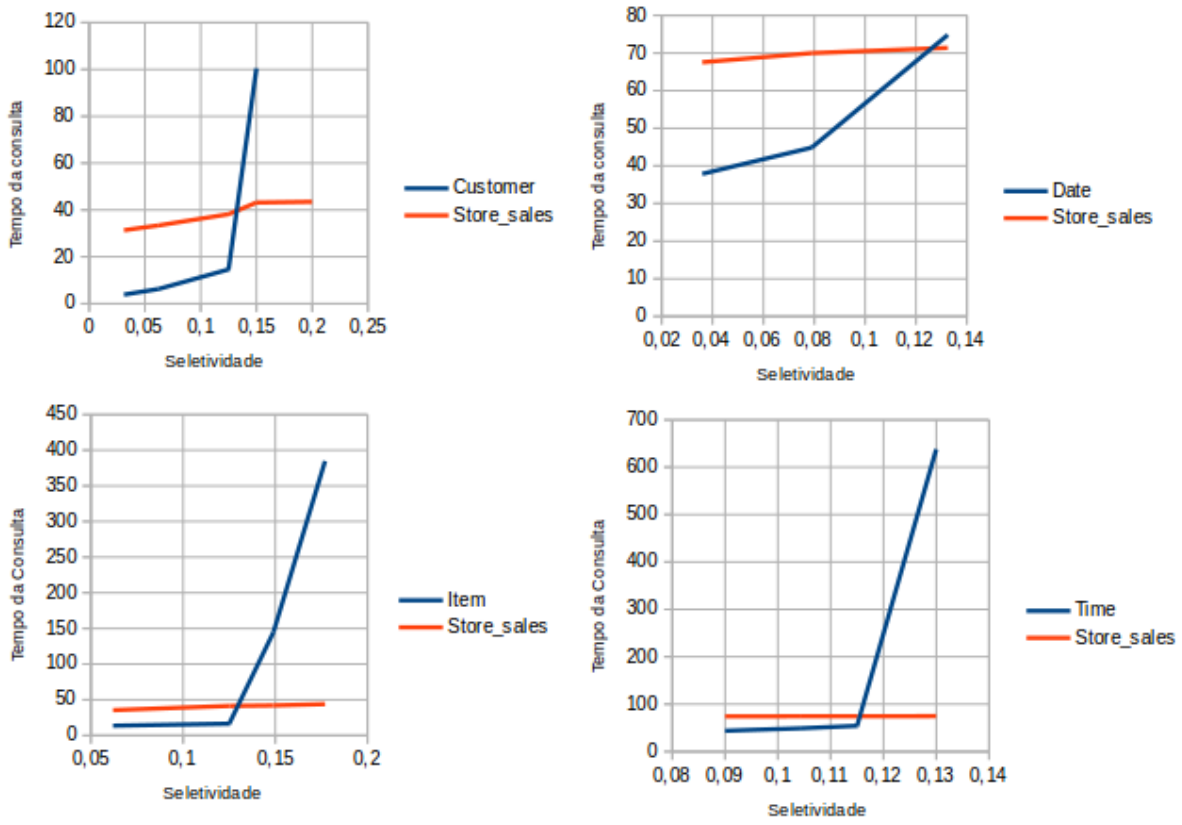

FIG. 5.4: Gráficos Tempo x Seletividade no OrientDB utilizando documentos aninhados.

Baseado nos experimentos tanto no OrientDB utilizando dados aninhados, quanto no MongoDB, pode-se notar que a seletividade de corte das coleções de um mesmo conjunto de dados eram próximas. Portanto, como uma solução para melhorar o desempenho do OrientDB recomenda-se que após a construção dos agregados utilizando a heurística com uma seletividade de corte mais baixa, seja obtida uma nova seletividade de corte a partir das coleções construídas e que se reaplique a heurística.

No caso do TPC-DS, se após construir a coleção *Item* e *Store_Sales*, fossem realizados testes de forma a obter um novo valor para a seletividade de corte, o valor de 0,125 seria encontrado. Este valor poderia ser utilizado para reaplicar a heurística e obter novas modelagens que iriam favorecer as consultas, antes menos favorecidas.

Alternativamente, uma maneira mais simples de obter um novo valor para a seletivi-

dade de corte, seria executar a consulta 89 sobre a coleção *Item*. A escolha desta consulta nesta coleção é devida ao fato de que a mesma possui a SCR (0,112) mais próxima da seletividade de corte (0,05), mas ainda maior que esta. Ao observar que esta consulta permite um melhor desempenho na coleção *Item* do que na coleção *Store_Sales*, a seletividade de corte pode ser reconsiderada, tomando-se o valor 0,112. A tabela 5.9 apresenta o conjunto de coleções que seriam selecionadas ao reaplicar a heurística utilizando o valor de corte de 0,125 ou 0,112 obtida pela consulta 89.

TAB. 5.9: Seletividade dos campos raízes das coleções do TPC-DS, e no final, o número de documentos em cada coleção de agregado.

Consultas	Store_Sales	Customer	Date	Item	Time
3			0,2	0,0008	
6			0,2	0,16	
19		0,0009	0,032	0,019	
27	0,99	0,013	0,2		
53			0,25	0,303	
68	0,833	0,25	0,04		
71			0,032	0,017	0,5
79		0,387	0,084		
88		0,27			0,019
89			0,201	0,112	
96		0,09			0,041
98			0,0121	0,303	
Documents	2652870	100000	73049	18000	86400

TAB. 5.10: Média dos tempos de execução das consultas do TPC-DS no OrientDB em milissegundos.

Consulta	Store_Sales	Customer	Date	Item	Time	Ganho
3	61465	86275	116757	25979	111247	57%
6	73699			45219		38%
19	74186	28262	32457	26174	40253	61%
27	78887	28938	517967		50258	63%
53	61465					0%
68	80450	32962	35814			55%
71	68697	37864	38903	27873	40147	59%
79	83066	43391	47384			42%
88	68581	31247			32543	0%
89	64584			24228		62%
96	61465	29627			32706	65%
98	83066	121873	33131	102562		60%
Média	74695	48937	117487	42005	51192	47%

Ao seguir a heurística para o novo valor da seletividade de corte as coleções *Customer*

e *Date* seriam mantidas e o ganho obtido passaria a ser de 47%, conforme a tabela 5.10.

Após a correção do valor da seletividade de corte, algumas consultas que estariam associadas às modelagens escolhidas obtiveram um melhor desempenho em outras coleções. Porém, os tempos de execução das consultas nas coleções associadas foram bem próximos aos melhores tempos, com exceção da consulta 88 executada sobre a coleção *Store_Sales*. O motivo pelo qual a consulta 88 foi uma exceção, é o mesmo já explicado na seção 5.6.

Portanto, apesar do valor de corte escolhido inicialmente não ter permitido escolher um conjunto de coleções que permitisse uma melhoria considerável nas consultas, pode-se considerar que a heurística proposta para o OrientDB ainda é válida, pois mesmo assim provê algum ganho. Mas vale ressaltar que a mesma pode ser reaplicada, reconsiderando o valor da seletividade de corte após a construção das primeiras coleções.

5.8.2 VALIDAÇÃO DA HEURÍSTICA DO ORIENTDB UTILIZANDO DOCUMENTOS LIGADOS

Para validarmos a heurística do OrientDB utilizando documentos ligados foram utilizados os valores das SCR das coleções do TPC-DS da tabela 5.11 e o valor de seletividade de corte de 0,2.

TAB. 5.11: Seletividade dos campos raízes das coleções do TPC-DS, e no final, o número de documentos em cada coleção de agregado.

Consultas	Store_Sales	Customer	Date	Item	Time
3			0,2	0,0008	
6			0,2	0,16	
19		0,0009	0,032	0,019	
27	0,99	0,013	0,2		
53			0,25	0,303	
68	0,833	0,25	0,04		
71			0,032	0,017	0,5
79		0,387	0,084		
88		0,27			0,019
89			0,201	0,112	
96		0,09			0,041
98			0,0121	0,303	
Documentos	2652870	100000	73049	18000	86400

Passo (i):

Neste passo o conjunto R será definido como um conjunto vazio.

Passo (ii):

Os seguintes conjuntos serão formados se considerarmos que o conjunto C^D está ordenado segundo a tabela 5.11, onde o primeiro elemento é a coleção *Customer*:

Coleção *Customer*:

- $Q_L^{Customer} = \{19, 27, 96\}$;
- $Q^{Customer} = \{19, 27\}$;
- $Q^A = \{19, 27\}$;
- $R = \{C^{Customer}\}$;

Coleção *Date*:

- $Q_L^{Date} = \{3, 6, 19, 27, 68, 71, 79, 98\}$;
- $Q^{Date} = \{68, 79, 98\}$;
- $Q^A = \{19, 27, 68, 79, 98\}$;
- $R = \{C^{Customer}, C^{Date}\}$;

Coleção *Item*:

- $Q_L^{Item} = \{3, 6, 19, 71, 89\}$;
- $Q^{Item} = \{3, 6, 71, 89\}$;
- $Q^A = \{19, 27, 68, 79, 98, 3, 6, 71, 89\}$;
- $R = \{C^{Customer}, C^{Date}, C^{Item}\}$;

Coleção *Time*:

- $Q_L^{Time} = \{88, 96\}$;
- $Q^{Time} = \{88, 96\}$;
- $Q^A = \{19, 27, 68, 79, 98, 3, 6, 71, 89, 88, 96\}$;
- $R = \{C^{Customer}, C^{Date}, C^{Item}, C^{Time}\}$;

Passo (iii):

Inicialmente o conjunto R será ordenado da seguinte forma:

- $R = \{C^{Time}, C^{Customer}, C^{Item}, C^{Date}\}$;

Como $|Q^{Time}|/|Q| < 0,2$, a coleção *Time* será descartada de R. Como dentre as consultas de Q^{Time}, excluídas de Q^A, a consulta 96 possui SCR menor que a de corte na coleção *Customer*, esta consulta será associada a esta coleção e re-adicionada a Q^A. Porém, a consulta 88 é retirada e fica órfã. Logo, temos que:

- $Q^{Customer} = \{19, 27, 96\}$
- $Q^A = \{19, 27, 68, 79, 98, 3, 6, 71, 89, 96\}$;

Após a inclusão da consulta 96 ao conjunto de consultas associadas à coleção *Customer*, esta passou a favorecer mais do que 20% das consultas ($|Q^{Customer}|/|Q| > 0,2$) e por isso não será descartada.

Como $|Q^x|/|Q| > 0,2$ em todas as outras coleções $x \in R$, nenhuma outra será descartada.

Passo (iv): As consultas órfãs restantes, após a escolha das modelagens são as consultas 53 e 88.

Como a coleção *Customer* possui a menor SCR na consulta 88, esta consulta é associada a ela. E a consulta 53 é associada à coleção *Date* pelo mesmo motivo. Assim, temos:

- $Q^{Customer} = \{19, 27, 96, 88\}$
- $Q^{Date} = \{68, 79, 98, 53\}$.

Passo (v):

Como $R \neq \{\}$, a coleção C^F não é adicionada.

Ao final, após seguir todos os passos da Heurística OL1, as coleções *Item*, *Customer* e *Date* são escolhidas para melhorar o desempenho das consultas, e as consultas estão associadas às coleções da seguinte forma:

- $Q^{Customer} = \{19, 27, 96, 88\}$
- $Q^{Date} = \{68, 79, 98, 53\}$;
- $Q^{Item} = \{3, 6, 71, 89\}$;

Ao analisarmos a tabela 5.12 podemos observar que o ganho obtido ao utilizar a heurística foi de 73%, e que além das duas consultas que eram melhor favorecidas pela

coleção *Time* retirada do conjunto R, somente uma consulta obteve o segundo melhor tempo na coleção associada. Para as demais consultas, a coleção associada obteve o melhor tempo.

TAB. 5.12: Média dos tempos de execução das consultas do TPC-DS no OrientDB em milissegundos.

Consulta	Store_Sales	Customer	Date	Item	Time	Ganho
3	53200	45680[a]	47740	1280	51470[a]	98%
6	56530	552961[b]	52740	15827	55350[a]	73%
19	70020	2066	5210	2400	46500[a]	98%
27	71060	3073	42380	77194[b]	52950[a]	96%
53	76210	75000[a]	69510[b]	32293	76690[a]	9%
68	79100	15040	11801	80820[a]	80110[a]	86%
71	71510	47780[a]	11690	2686	22230	97%
79	84000	25370	23164	85540[a]	85130[a]	73%
88	60060	72890[b]	82620[a]	81870[a]	2821	-17%
89	81660	80610[a]	50900	12211	81780[a]	86%
96	60490	5660	71090[a]	71220[a]	4004	94%
98	71210	66760[a]	5168	28920	68380[a]	93%
Média	69587	41268	39501	41021	52284	73%

Apesar do ganho obtido ao aplicar a heurística utilizando uma seletividade de corte no valor de 0,2, ter sido satisfatório, foram realizados testes para encontrar um valor de corte mais exato para as coleções, de forma a averiguar se seria possível aumentar ainda mais o ganho.

Nesses testes, para valores de SCR maiores que 0,5 o erro GCOLE ocorreu. Portanto, somente foi possível obter um valor de corte para as coleções onde este valor era menor que 0,5.

Nos gráficos da figura 5.5, podemos visualizar que somente na coleção *Date* a seletividade de corte foi encontrada, com um valor de 0,32 aproximadamente. Nas outras coleções, não foi possível obtê-lo. Portanto, o valor de 0,5 foi considerado como o valor da seletividade para essas coleções.

A heurística proposta não tem como objetivo redirecionar ou reescrever consultas, porém vale discutir o caso das consultas 53 e 88, que foram associadas às coleções *Date* e *Customer*, respectivamente. Como estas coleções foram geradas utilizando relacionamento reverso, estas consultas podem ser re-escritas para usá-los ou não. No entanto, para melhor desempenho dessas consultas, é melhor não utilizá-los. Foram realizados testes de forma

[a]Consulta realizada diretamente sobre a classe store_sales
[b]Consulta realizada diretamente sobre a classe store_sales utilizando relacionamento reverso

a obter o tempo de execução dessas consultas sem utilizar relacionamentos reversos. Na consulta 53 sobre a coleção *Date* foi encontrado o tempo de 49,43 segundos e na consulta 88 sobre a coleção *Customer* foi encontrado o tempo de 12,18 segundos. Verificou-se então, que estas consultas seriam mais rápidas, se realizadas sem utilizar relacionamentos reversos, com ganho médio final de 84%.

Portanto, analisando os resultados dos experimentos de validação da heurística, pode-se concluir que a heurística proposta é válida, e que o ganho inicial obtido pela aplicação da heurística dependerá da proximidade entre o valor de corte escolhido e o seu valor real. Este valor pode ser obtido experimentalmente, de forma a aumentar ainda mais o ganho pela reaplicação da heurística proposta.

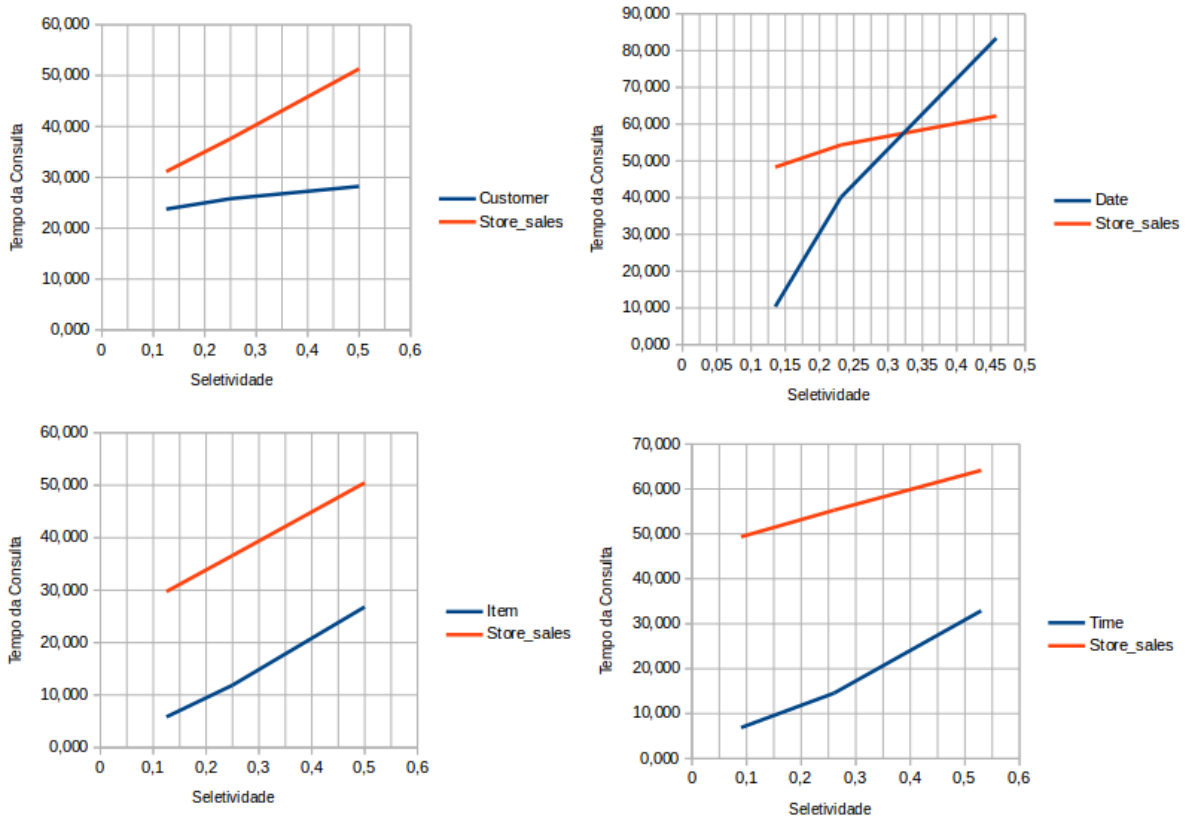

FIG. 5.5: Gráficos Tempo x Seletividade no OrientDB utilizando documentos ligados.

5.9 VALIDAÇÃO DO COMPORTAMENTO DO COUCHDB

A partir da tabela 5.13 é possível observar que o tempo de criação das visões materializadas em cada coleção teve um tempo aproximado, independente da consulta que gerou a visão.

A partir dos tempos da tabela 5.14 podemos observar que os tempos da consulta às visões foram próximos na maioria das consultas. Nas consultas 53, 68, 79 e 98, em

algumas coleções os tempos foram um pouco mais baixos, porém a diferença de tempo não teve relação alguma com os atributos utilizados nas consultas. Na consulta 79, por exemplo, não havia nenhum atributo da raiz dos documentos da coleção *Time* como filtro ou na cláusula de projeção, porém esta coleção foi onde esta consulta teve um de seus melhores desempenhos.

TAB. 5.13: Tempo de construção das visões materializadas das consultas do TPC-DS.

Visão	Store_Sales	Customer	Date	Item	Time
3	10:20,204	5:33,831	9:10,174	5:15,737	6:10,342
6	10:22,761	5:32,738	9:13,62	5:15,649	6:10,666
19	10:22,306	5:32,806	9:12,897	5:15,378	6:08,088
27	10:24,080	5:32,955	9:12,566	5:17,831	6:09,648
53	10:25,841	5:48,084	9:15,115	5:15,974	6:21,818
68	10:24,644	5:32,963	9:10,575	5:18,213	6:09,831
71	10:22,610	5:35,114	9:10,471	5:14,999	6:07,973
79	10:22,453	5:33,201	9:11,122	5:20,830	6:11,108
88	10:17,433	5:33,873	9:10,793	5:18,912	6:05,322
89	10:24,427	5:26,742	9:09,429	5:16,580	6:10,987
96	10:18,060	5:32,783	9:12,589	5:17,147	6:04,321
98	10:23,184	5:29,290	9:10,002	5:16,931	6:09,555

TAB. 5.14: Média do tempo de execução das consultas às visões do TPC-DS.

Consulta	Store_Sales	Customer	Date	Item	Time
3	13	14	11	9	12
6	24	25	21	22	22
19	7	8	7	7	7
27	11	12	11	10	11
53	843	881	814	187	849
68	205	88	90	191	93
71	30	31	28	32	28
79	525	229	222	525	246
88	7	9	7	9	8
89	250	248	208	227	243
96	8	9	8	8	9
98	207	209	200	125	205

Ainda na direção de tentar entender o motivo da diferença de tempos, foram medidos os tamanhos da visões materializadas geradas nas coleções *Customer* e *Store_Sales* para a realização da consulta 79, obtendo 973 KB e 7622 KB, respectivamente. Notou-se que os espaços ocupados pelas visões geradas foram diferentes, mesmo quando estas visões eram equivalentes, i.e., para atender a mesma consulta (79).

Foi realizado um estudo para averiguar se os tamanhos das visões estavam relacionados com os atributos utilizados na cláusula de projeção das consultas, tendo em vista que na consulta 53 havia somente atributos da raiz da coleção *Item* na cláusula de projeção e esta consulta foi mais rápida nesta coleção. Porém, nas consultas 79 e 89 não havia nenhum atributo da raiz da coleção *Date* nas cláusulas de projeção dessas consultas e esta coleção foi a mais rápida em ambas, contrariando a hipótese acima levantada.

Além disso, foram realizadas consultas e pesquisas em fóruns de discussão que tratavam sobre CouchDB. A reposta encontrada mencionava que a forma na qual a função utilizada para a criação da visão materializada poderia estar causando este tamanho diferente. Porém, a forma como a função foi usada em todas coleções foi igual, e por isso esta hipótese também foi descartada.

Portanto, após diversos estudos e pesquisas não foi encontrado, neste trabalho, o motivo pelo qual as duas visões resultaram em espaço de armazenamento diferente, e que consequentemente, resultou em tempos de consulta diferentes para essas coleções.

A partir dos experimentos realizados no CouchDB, embora tenham ocorrido tempos de execução diferentes, para a mesma consulta, em coleções distintas, pode-se confirmar que a escolha de uma ou outra modelagem de agregado neste SGBD seria pouco vantajosa, e que em virtude dos tempos de execução de todas as consultas às visões materializadas terem sido abaixo de 1 segundo, com diferenças insignificantes, os aspectos a serem considerados para a escolha das modelagens de agregados devem ser outros, como por exemplo o espaço de armazenamento.

5.10 VALIDAÇÃO DO COMPORTAMENTO DO MARKLOGIC

Da mesma forma que no capítulo anterior, os tempos de execução das consultas do MarkLogic não serão apresentados e a validação do seu comportamento será realizada somente por meio da tabela de seletividade dos campos raízes das coleções.

A partir da análise da tabela 5.15 que apresenta somente a coleção onde uma consulta foi mais rápida e a SCR das coleções, é possível notar que da mesma forma que no SSB não houve um padrão de comportamento que pudesse levar a uma criação de uma heurística para este SGBD.

A coleção *Date*, por exemplo, possui uma SCR igual a 0,04 na consulta 68, porém o tempo de execução desta consulta nesta coleção foi o segundo pior de todos, mesmo tendo campos com a menor seletividade na consulta. Outra consulta que apresentou um comportamento cuja SCR não exerceu nenhuma influência foi a consulta 96, pois, embora

a coleção *Item* possua uma SCR (0,303) muito acima da SCR da coleção *Date* (0,0121) a consulta naquela coleção foi muito melhor.

Logo, assim como nos resultados do SSB não foi encontrado no conjunto de dados e consultas do TPC-DS um comportamento padrão a partir do qual se possa extrair uma heurística de modelagem de agregados, em benefício de consultas analíticas.

TAB. 5.15: Seletividade dos campos raízes das coleções do TPC-DS, e no final, o número de documentos em cada coleção de agregado.

Consultas	Store_Sales	Customer	Date	Item	Time
3			0,2	0,0008	
6			0,2	0,16	
19		0,0009	0,032	0,019	
27	0,99	0,013	0,2		
53			0,25	0,303	
68	0,833	0,25	0,04		
71			0,032	0,017	0,5
79		0,387	0,084		
88		0,27			0,019
89			0,201	0,112	
96		0,09			0,041
98			0,0121	0,303	
Documentos	2652870	100000	73049	18000	86400

6 IMPACTO DA MODELAGEM DE AGREGADOS UTILIZANDO ÍNDICES

No capítulo 4, o estudo apresentado levou em consideração a configuração padrão de cada SGBD, e devido à diversidade de abordagens destes, não foi explorada a utilização de índices na maioria dos SGBD estudados. No caso do ElasticSearch, em sua configuração padrão, índices sobre todos os campos são criados automaticamente. Não é permitida qualquer interferência do modelador quanto a não criar índices ou quanto a criar novos tipos de índices. Portanto, o estudo do impacto da variação da modelagem dos agregados ao utilizar índices desse SGBD já foi realizado nos capítulos 4 e 5. Já o CouchDB, por realizar consultas sobre sobre visões materializadas não necessita de índices para as consultas. Portanto, este capítulo discute o impacto da variação da modelagem dos agregados somente para os SGBD MongoDB, OrientDB e MarkLogic, para as consultas e dados do TPC-DS.

Da mesma forma descrita nos capítulos 4 e 5, quando os índices não eram utilizados, a análise do comportamento somente será feita sobre as consultas realizadas sequencialmente, excluídos os tempos da primeira execução das consultas aos bancos de dados.

Diferentemente daquele capítulo, não serão propostas heurísticas de modelagens de agregados para os SGBD utilizando recurso de índices, pois não seria viável dentro do tempo disponível para a realização deste trabalho.

6.1 ÍNDICES NO MONGODB

Como mencionado no capítulo 2, no MongoDB é possível a utilização de índices compostos e índices de campos individuais. Nos testes realizados, as consultas foram executadas primeiramente com a disponibilidade de índices individuais, criados com base nos campos usados nos filtros das consultas. A seguir, estes índices foram descartados, e foram criados índices compostos, seguindo o mesmo critério anterior. As consultas foram re-executadas, e o melhor tempo de cada consulta foi considerado.

A tabela 6.1 mostra os resultados das consultas do TPC-DS no MongoDB utilizando tanto índices de campos individuais como índices compostos, onde as consultas que tiveram melhor desempenho utilizando índices compostos estão sinalizadas com a letra ”c” sobrescrita.

A partir dos resultados obtidos é possível observar que a coleção *Store_Sales* obteve um melhor desempenho em todas as consultas. Esse comportamento diferente, quando utilizando índices, se deve ao fato de que ao utilizar índices na coleção *Store_Sales*, somente os documentos que satisfazem à condição de seleção são trazidos para a memória, não sendo necessário varrer todos os documentos, como era realizado nas consultas sem índices, nem realizar a operação de *unwind* nesta coleção. Já nas outras coleções, esta operação ainda é necessária. Portanto, a coleção baseada na tabela de fatos, quando utilizando índices, permite uma melhor performance do que as coleções baseadas nas tabelas de dimensão.

TAB. 6.1: Média dos tempos de execução consultas do TPC-DS no MongoDB utilizando índices.

Consulta	Store_Sales	Customer	Date	Item	Time	Ganho
3	70	682[c]	5700[c]	78	1692[c]	0%
6	1935,2	14191[c]	7947[c]	3422[c]	24557[c]	0%
19	66	82[c]	405[c]	242[c]	299	0%
27	361,6	435[c]	8028[c]	18179[c]	14512	0%
53	3620,6	26591[c]	9647[c]	8375	35214	0%
68	1148[c]	2042	2052[c]	31330[c]	12684	0%
71	334,8	4814[c]	1351[c]	572[c]	7196	0%
79	2210[c]	5944	4540[c]	28044[c]	24295	0%
88	368[c]	744	43250[c]	35419[c]	471[c]	0%
89	2342,2	17236[c]	9496[c]	3450	27321[c]	0%
96	237	475[c]	38475[c]	28887[c]	624[c]	0%
98	712	14942[c]	985	5918	25433[c]	0%
Média	1117	7348	10990	13660	14525	0%

Ao se analisar o aumento do desempenho obtido utilizando índices no MongoDB, e comparando os tempos das tabelas 6.1 e 5.3, foi possível observar um ganho médio de 34% nas coleções que mais favoreceram as consultas e um ganho médio de 17% levando em consideração o tempo de todas as consultas.

6.2 ÍNDICES NO ORIENTDB

Devido ao melhor desempenho do OrientDB utilizando documentos ligados, neste capítulo a análise do comportamento do OrientDB utilizando índices somente será feita nesta forma de organizar os dados.

Conforme mencionado no capítulo 2, apesar de ser possível criar índices em campos aninhados, ao organizar os dados utilizando documentos ligados, o OrientDB somente

[c]Índice composto

reconhece os índices existentes na classe onde a consulta foi realizada. Portanto, os únicos índices que podem ser utilizados no OrientDB são os índices criados para os campos da classe que representa os dados das raízes dos documentos.

No OrientDB também é possível criar índices compostos nos campos das raízes dos documentos, porém, para utilizá-los é necessário reescrever a consulta, de forma a indicar qual índice deve ser utilizado.

A consulta abaixo, é realizada utilizando um índice composto, criado sobre três campos, *cd_gender*, *cd_marital_status* e *cd_education_status*, onde para cada combinação de valores é criado um índice, como se cada combinação fosse uma cadeia de caracteres.

```
SELECT rid.store_sales as store FROM index:gme
WHERE key=["F","S","Secondary"]
```

Nos experimentos, se era possível utilizar índices compostos em uma consulta estes foram utilizado, pois diferentemente do MongoDB, o OrientDB não faz interseção de índices. Logo, uma consulta utilizando índices compostos será sempre mais eficiente do que uma consulta utilizando somente um campo individual.

Na tabela 6.2 são apresentados os tempos das consultas sobre as coleções, onde quando foi possível utilizar algum índice, o tempo está sinalizado com a letra ”i” sobrescrita. O ganho apresentado na última coluna é o ganho de tempo obtido pela realização da consulta na modelagem que melhor a favoreceu, com relação ao tempo da mesma consulta na coleção *Store_Sales*, independentemente se foi possível utilizar índices nesta coleção.

TAB. 6.2: Média dos tempos de execução consultas do TPC-DS no OrientDB utilizando índices.

Consulta	Customer	Date	Item	Time	Store_Sales	Ganho
3	45679	46284^{i}	840^{i}	51469	53204	98%
6	109131	39615,2^{i}	13836^{i}	55347	56534	76%
19	754^{i}	8621^{i}	1983,4^{i}	46502	70020	99%
27	1693^{i}	38755,4^{i}	188692	52952	65351^{i}	97%
53	75002	48565,6^{i}	32209^{i}	76688	76208	58%
68	22432,2^{i}	9807^{i}	80820	80111	63946^{i}	85%
71	47775	8922,8^{i}	2201^{i}	29212,2^{i}	71510	97%
79	31600^{i}	19639^{i}	85537	85125	84004	77%
88	18160^{i}	82621	81871	2115^{i}	9673^{i}	78%
89	80605	44162,8^{i}	11292^{i}	81778	81657	86%
96	6369,6^{i}	71088	71216	3302^{i}	9164^{i}	64%
98	66756	3857^{i}	29610^{i}	68377	71210	95%
Média	42163	35162	50009	52748	59373	84%

O impacto da variação da modelagem dos agregados no OrientDB utilizando índices foi similar ao impacto sem utilizar índices, pois diferentemente do MongoDB, não é possível utilizar índices criados em campos aninhados. Isso fez com que a coleção *Store_Sales* não pudesse se aproveitar dos índices criados em campos que não estivessem nas raízes dos seus documentos, e se valesse da possibilidade de resolver as consultas sem realizar a operação *unwind*. Logo, essa modelagem de agregado no OrientDB continuou a ser a pior de todas. Portanto, a SCR no OrientDB, utilizando índices continuou influenciando no desempenho das coleções nas consultas, onde a coleção com a menor SCR foi a que obteve o melhor desempenho em cada consulta.

Comparando o desempenho entre os tempos das consultas utilizando índices da tabela 6.2 com os tempos das consultas sem utilizar índices da tabela 5.8, os ganhos obtidos ao utilizar índices não foram muito significativos, onde os tempos nas coleções que mais favoreceram as consultas obtiveram um ganho de apenas 12% e o ganho médio obtido em todas as consultas que puderam utilizar índices foi de apenas 22%.

6.3 ÍNDICES NO MARKLOGIC

Da mesma forma que no capítulo 4 e 5, os tempos de execução das consultas no MarkLogic foram omitidos e o comportamento deste SGBD se dará a partir de uma tabela que somente mostrará a coleção onde uma consulta foi mais rápida.

Conforme mencionado no capítulo 2, além dos índices criados automaticamente pelo MarkLogic, é possível criar outros dois tipos de índices neste SGBD que servem para otimizar consultas sobre intervalos de valor. Estes índices são conhecidos no MarkLogic como *element-range-index* e *element-path-range-index*. O segundo tipo é utilizado quando mais de um atributo em níveis diferentes da estrutura dos documentos possuem o mesmo nome e é necessário indexar somente um campo entre os que possuem o mesmo nome, sendo necessário definir a localização correta do campo que será indexado.

Assim, foram criados índices individuais sobre os campos utilizados nas cláusulas de seleção das consultas e para a coleção baseada na tabela de fatos também foram criados índices para os campos utilizados na cláusula de projeção, como será explicado mais adiante.

Apesar de ser possível a utilização de índices do tipo *element-range-index* nas coleções baseadas nas tabelas de dimensão, em testes iniciais verificou-se que o tempo de execução das consultas nessas coleções, utilizando este tipo de índice, foi pior do que quando os

[i]Consulta onde foi possível utilizar índices

índices do tipo *element-path-range-index* foram utilizados. A consulta 3 na coleção *Item*, por exemplo, foi mais rápida utilizando *element-range-index* do que utilizando *element-path-range-index*. Logo, nas consultas sobre as coleções baseadas nas tabelas de dimensão, somente foram utilizados índices do tipo *element-path-range-index*.

Entretanto, na coleção baseada na tabela de fatos, *Store_Sales*, os índices do tipo *element-range-index* foram mais eficientes que os índices *element-path-range-index*. Ao criar os índices do tipo *element-range-index*, é possível que estes índices sejam indicados na própria consulta. Desta forma, pode-se forçar a sua utilização, inclusive na cláusula de projeção. Na tabela 6.3 é possível notar que a coleção *Store_Sales* obteve um melhor desempenho que as demais. O motivo provável para que isto tenha ocorrido é a utilização dos índices do tipo *element-range-index*, indicados na cláusula de projeção da consulta. Um exemplo deste tipo de consulta é apresentado a seguir.

TAB. 6.3: Média dos tempos de execução consultas do TPC-DS no MarkLogic utilizando índices em milisegundos.

Consultas	Store_Sales	Customer	Item	Time	Date
3					
6					
19					
27					
53					
68					
71					
79					
88					
89					
96					
98					

Para utilização dos índices de intervalo de ambos os tipos foi necessário reescrever a consulta, indicando explicitamente onde utilizar esses recursos (cts:path-range-query). A consulta 53 do TPC-DS reescrita para utilizar índices de intervalo do tipo *element-path-range-index* sobre a coleção *Customer* pode ser visualizada a seguir.

```
<query53>{
 let $m := map:map()
  let $build :=
for $sales in cts:search(doc()/store_sales,
cts:and-query((
cts:path-range-query("/store_sales/item/i_category", "=",
("Music","Home","Sports")),
cts:path-range-query("/store_sales/date/
d_month_seq", ">=", 1200),
```

```
cts:path-range-query("/store_sales/date/
d_month_seq", "<=", 1211))))
    let $i_item_id:= $sales/item/i_item_id
    let $i_item_desc:= $sales/item/i_item_desc
    let $i_category:= $sales/item/i_category
    let $i_class:= $sales/item/i_class
    let $i_current_price:= $sales/item/i_current_price
    let $key := concat($i_item_id, ',',
    $i_item_desc, ',', $i_category, ',',
    $i_class, ',', $i_current_price)
    return map:put(
      $m, $key, sum((
        map:get($m, $key),
        xs:float($sales/ss_ext_sales_price))))
  for $key in map:keys($m)
  return
  <group>{concat($key,',',map:get($m, $key))}</group>
}</query53>
```

A mesma consulta executada sobre a coleção *Store_Sales*, porém utilizando o tipo de índice *element-range-index* também pode ser visualizada abaixo, onde os termos *element-range-query* e *element-reference* forçam a utilização deste tipo de índice nas cláusulas de seleção e projeção, respectivamente:

```
<query53>{
let $m := map:map()
let $build :=
let $q:=cts:and-query((
cts:element-range-query(xs:QName("i_category"), "=",
("Music","Home","Sports")),
cts:element-range-query(xs:QName("d_month_seq"), ">=", 1200),
cts:element-range-query(xs:QName("d_month_seq"), "<=", 1211)
))
for $tuple in cts:value-tuples(
(
cts:element-reference(xs:QName('i_item_id')),
cts:element-reference(xs:QName('i_item_desc')),
cts:element-reference(xs:QName('i_category')),
cts:element-reference(xs:QName('i_class')),
cts:element-reference(xs:QName('i_current_price')),
cts:element-reference(xs:QName('ss_ext_sales_price'))
),
("item-frequency", "frequency-order"),
$q)
let $key := concat($tuple[1], ',',
$tuple[2], ',', $tuple[3], ',', $tuple[4], ',', $tuple[5])
    return map:put(
      $m, $key, sum((
        map:get($m, $key),xs:long($tuple[6]))))
  for $key in map:keys($m)
  return
  <group>{concat($key,',',map:get($m, $key))}</group>
}</query53>
```

Uma outra alternativa da indicação de índices na consulta foi feita, combinando o tipo de índice *element-path-range-index* na cláusula de seleção e o tipo de índice *element-range-index* na cláusula de projeção. Porém, o tempo das consultas não melhorou. Assim, foi descartado esta forma de uso de índices.

Ao analisar o aumento do desempenho do MarkLogic ao utilizar os índices de intervalo (omitido), houve um ganho considerável na coleção baseada na tabela de fatos (Store_Sales), porém nas coleções baseadas nas tabelas de dimensão houve uma queda no desempenho das consultas. Portanto, a utilização desse tipo de índice nestas coleções deve ser evitado. Por outro lado, o uso desse tipo de índice sobre as coleções baseadas na tabela de fatos pode ser recomendados.

7 COMPARAÇÃO DE DESEMPENHO ENTRE OS SGBD

Este capítulo faz uma comparação de desempenho entre os SGBD orientados a documentos estudados. No entanto, como estes possuem características ou abordagens diferentes, em especial no que diz respeito à utilização de índices, não seria justo comparar os SGBD que criam índices automaticamente com os que não os criam. Por isso, duas comparações de desempenho foram realizadas, uma entre os SGBD que não criam índices automaticamente e outra entre os SGBD utilizando os índices da melhor forma possível para cada SGBD.

Dentre os SGBD estudados, somente o MongoDB e o OrientDB permitem a realização de consultas sem a utilização de índices, portanto a comparação entre os SGBD sem a utilização de índices somente será feita entre esses dois sistemas.

Devido ao fato do CouchDB utilizar somente o recurso de visões materializadas para a realização das suas consultas, o desempenho deste banco não foi comparado com os demais.

A comparação de desempenho do MarkLogic com os demais SGBD, apesar de ter sido realizada, não será apresentada, devido à limitação da sua licença de uso.

7.1 COMPARAÇÃO DE DESEMPENHO SEM A UTILIZAÇÃO DE ÍNDICES

A comparação de desempenho entre o MongoDB e o OrientDB, sem utilização de índices foi realizada levando em consideração as duas formas de organizar os dados no OrientDB, utilizando documentos aninhados e documentos ligados. Esta comparação foi realizada com base no desempenho das consultas sobre as coleções geradas, a partir dos dados do SSB e do TPC-DS.

7.1.1 MONGODB X ORIENTDB UTILIZANDO DOCUMENTOS ANINHADOS

7.1.1.1 CONSULTAS DO SSB

Nas tabelas 7.1, 4.2, 7.2 e 4.7 são apresentados os tempos da primeira execução das consultas do SSB e das consultas subsequentes nas diferentes coleções no MongoDB e no OrientDB utilizando documentos aninhados, respectivamente.

Ao comparar o tempo de execução de todas as consultas do SSB, o MongoDB foi mais rápido em todas as consultas que o OrientDB quando utilizado com documentos

TAB. 7.1: Tempos das primeiras execuções das consultas do SSB no MongoDB em milissegundos.

Consultas	Lineorder	Customer	Orderdate	Part	Supplier	Ganho
1.1	62179	121861	47846	113016	162140	23%
1.2	63193	90592	34988	68279	159460	45%
1.3	64025	104020	34355	78803	158940	46%
2.1	65785	119147	142253	43919	69144	33%
2.2	67654	120690	140946	41466	73374	39%
2.3	66166	58797	134776	41542	69929	37%
3.1	71007	51771	136322	130037	77153	27%
3.2	68162	34018	130487	87740	52289	50%
3.3	71647	30366	138940	53993	48588	58%
3.4	72641	29679	37100	52345	48079	59%
4.1	74427	49695	141940	72741	69449	33%
4.2	75522	49520	57818	74041	68987	34%
4.3	74762	48557	57675	48887	52153	35%
Média	69013	69901	95034	69755	85361	40%

TAB. 7.2: Tempos das primeiras execuções das consultas do SSB no OrientDB utilizando documentos aninhados em milissegundos.

Consulta	Lineorder	Customer	Orderdate	Part	Supplier	Ganho
1.1	261630					0%
1.2	249495		194178			22%
1.3	258483		192882			25%
2.1	227975			200645		11%
2.2	291703			176987		39%
2.3	224727			180865		19%
3.1	241382					0%
3.2	244251	151358			308596	38%
3.3	252187	196437			470598	22%
3.4	246787	198163	203039		481999	19%
4.1	239194					0%
4.2	238124					0%
4.3	244277			187199	311920	23%
Média	247708	181986	196699	186424	393278	17%

aninhados, sendo no total 3 vezes mais rápido na realização das primeiras consultas e 4,8 vezes mais rápido na realização das consultas subsequentes. Ao se comparar o tempo das consultas sobre as coleções que mais as favoreceram, o MongoDB foi cerca de 5 vezes mais rápido nas primeiras execuções das consultas e 18,5 vezes mais rápido nas consultas subsequentes.

7.1.1.2 CONSULTAS DO TPC-DS

Nas tabelas 5.3, 7.3, 5.8 e 7.4 são apresentados os tempos da primeira execução das consultas do TPC-DS e das consultas subsequentes nas diferentes coleções no MongoDB e no OrientDB utilizando documentos aninhados, respectivamente.

TAB. 7.3: Tempos das primeiras execuções das consultas do TPC-DS no MongoDB em milissegundos.

Consulta	Store_Sales	Customer	Item	Time	Date	Ganho
3	24326	14432	9389	18850	20194	61%
6	25351	26331	13461	38606	21429	47%
19	25442	12362	9175	17789	14449	64%
27	26522	12402	26333	29633	21060	53%
53	29047	39259	16976	48108	22910	41%
68	30371	14760	38017	30282	16788	51%
71	27765	17877	9419	23544	15529	66%
79	31004	19064	35954	39949	18825	39%
88	32889	13768	42453	17201	55711	58%
89	32940	32006	12820	42760	22723	61%
96	28448	13631	35318	17509	49443	52%
98	30566	28855	15069	40236	15193	51%
Média	28723	20396	22032	30372	24521	54%

TAB. 7.4: Tempos das primeiras execuções das consultas do TPC-DS no OrientDB utilizando documentos aninhados em milissegundos.

Consulta	Store_Sales	Customer	Date	Item	Time	Ganho
3	76793	101077	54218	40038	124227	48%
6	77611		194178	63327		19%
19	86031	52234	69170	40624	55247	53%
27	82774	52700	68471	200645	54753	37%
53	104024			176987		0%
68	90381	68894	76763	180865		24%
71	86337	61357	74634	41335	66004	53%
79	90918	76007	90573		308596	17%
88	75321	55899			55900	26%
89	98011	198163	203039	123567	481999	0%
96	79153	55961			55399	31%
98	96558	146173	68320	163275		30%
Média	86992	86846	99929	114518	150265	28%

Assim como nas consultas do SSB, o MongoDB foi superior em todas as consultas do TPC-DS em todas as coleções, sendo 3 vezes mais rápido nas primeiras consultas e 6 vezes mais rápido nas consultas subsequentes na média de todas as consultas. Considerando

somente os tempos das consultas sobre as coleções que mais as favoreciam, o MongoDB foi 5,1 vezes mais rápido nas primeiras consultas e 16,5 vezes mais rápido nas consultas subsequentes.

7.1.2 MONGODB X ORIENTDB UTILIZANDO DOCUMENTOS LIGADOS

7.1.2.1 CONSULTAS DO SSB

Nas tabelas 7.1, 4.2, 7.5 e 4.9 são apresentados os tempos de execução da primeira e das subsequentes execuções das consultas do SSB, nos SGBD MongoDB e no OrientDB utilizando documentos ligados, nas diferentes coleções.

Quando utilizando documentos ligados o OrientDB foi mais rápido que o MongoDB em 10 das 65 consultas quando executadas pela primeira vez nos bancos de dados, sendo 4,2 mais rápido nessas consultas. E em 8 das 65 consultas, quando executadas subsequentes à primeira o OrientDB foi 1,6 vezes mais rápido. Porém, no total de consultas realizadas o MongoDB foi 2,3 vezes mais rápido que o OrientDB nas primeiras execuções e 3,5 vezes mais rápido nas consultas subsequentes. No total de consultas sobre as coleções que mais as favoreceram, o MongoDB foi 1,7 vezes mais rápido nas primeiras execuções e 3,8 vezes mais rápido nas execuções subsequentes.

TAB. 7.5: Tempos das primeiras execuções das consultas do SSB no OrientDB, em documentos ligados em milissegundos.

Consulta	Lineorder	Customer	Orderdate	Part	Supplier	Ganho
1.1	264392	260702[a]	110012	151283[a]	244282[a]	58%
1.2	249192	284210[a]	10523	132030[a]	231498[a]	95%
1.3	253899	253739[a]	3533	139888[a]	228950[a]	98%
2.1	226166	219924[a]	155818[a]	113905[a]	148712	49%
2.2	296962	266592[a]	251917[a]	50986	274812	82%
2.3	224124	201524[a]	147098[a]	10789	211324	95%
3.1	244864	295801	201392[b]	238598[a]	296709	17%
3.2	192318	39386	192318[b]	210951[a]	36892	80%
3.3	253265	11869	209539[b]	198951[a]	7407	97%
3.4	243767	11566	10743	199806[a]	6798	97%
4.1	243911	155035	202642[a]	249780[b]	180310	36%
4.2	239533	150492	187633	226848[b]	180551	37%
4.3	240655	93027	320345	143418	51346	78%
Média	244080	172605	154116	159017	161507	71%

[a]Consulta realizada diretamente sobre a classe lineorder
[b]Consulta realizada utilizando relacionamento

7.1.2.2 CONSULTAS DO TPC-DS

A partir dos resultados do TPC-DS apresentados nas tabelas 5.3, 7.3 ,5.12 e 7.6 na média de todas as consultas, o MongoDB foi 3,4 vezes mais rápido na primeira consulta e 4,3 vezes mais rápido nas consultas subsequentes. Na média das consultas sobre as coleções que mais favoreceram as consultas o MongoDB foi 2 vezes mais rápido na primeira execução das consultas e 4,1 vezes mais rápido nas consultas subsequentes. Em sete consultas executadas pela primeira vez o OrientDB foi, na média dessas consultas, 2,2 vezes mais rápido que o MongoDB e em apenas duas consultas o OrientDB foi mais rápido que o MongoDB, quando a consulta foi realizada sequencialmente, obtendo um tempo de execução médio 1,45 vezes menor no OrientDB.

TAB. 7.6: Tempos das primeiras execuções das consultas do TPC-DS no OrientDB utilizando documentos ligados em milissegundos.

Consulta	Store_Sales	Customer	Date	Item	Time	Ganho
3	87421	209659[1]	54027	3281	101309[a]	97%
6	64697	291611[2]	58570	62356	136424[a]	10%
19	73697	8093	7685	14165	81833[a]	90%
27	75026	42874	69093	295983[b]	109793[a]	43%
53	85465	225027[a]	69198	66703	103096[a]	22%
68	78538	62004	19591	96019[a]	104990[a]	76%
71	73446	198030[a]	14733	13603	43065[a]	82%
79	78675	87320	41555	106637[a]	110358[a]	48%
88	69370	62728	91353[a]	93834[a]	4941	93%
89	83141	249050[a]	66324	45733	110231[a]	45%
96	68412	46637	90701[a]	84781[a]	7727	89%
98	83696	215035[a]	6797	61327	111558[a]	92%
Média	76798	141505	49135	78701	85443	66%

Tanto nas consultas do TPC-DS e do SSB, grande parte das consultas que tiveram melhor desempenho no OrientDB, utilizando documentos ligados, eram consultas cuja SCR era mais baixa, porém isto não ocorreu em todas as consultas, não sendo possível afirmar que para consultas em coleções com SCR baixo, o OrientDB, utilizando documentos ligados, é mais eficiente que o MongoDB.

7.2 COMPARAÇÃO DE DESEMPENHO UTILIZANDO ÍNDICES

Da mesma forma que seção anterior, por ocasião da comparação do desempenho dos SGBD sem utilizar índices, os tempos das primeiras execuções das consultas realizadas

[1]Consulta realizada diretamente sobre a classe lineorder
[2]Consulta realizada utilizando relacionamento

em cada banco de dados são apresentados e comparados.

Nas tabelas 7.7, 7.8, 7.9, 6.1, 5.6 e 6.2 são apresentados os tempos das consultas nos SGBD MongoDB, ElasticSearch e OrientDB respectivamente, utilizando índices nas coleções com diferentes modelagens de agregados, quando as consultas são realizadas pela primeira vez e a média de cinco execuções realizadas consecutivamente após a primeira execução.

As tabelas 7.10 e 7.11 apresentam um resumo dos resultados das tabelas acima citadas, onde são apresentadas as médias de todas as 60 consultas realizadas nos SGBD, nas diferentes coleções e a média dos melhores tempos obtidos por uma consulta em uma determinada coleção (consultas sobre coleções que melhor as favoreceram).

TAB. 7.7: Tempos das primeiras execuções das consultas do TPC-DS nas diferentes coleções no MongoDB em milissegundos, utilizando índices.

Consulta	Store_Sales	Customer	Item	Time	Date	Ganho
1	627	8130	442[c]	8835	7639[c]	30%
2	33003	25134	20488[c]	33374	12177	63%
3	5648	703[c]	1202[c]	2400	1678[c]	88%
4	13874[c]	2571[c]	23624	23765	11951[c]	81%
5	92620	217753	24195	162437	12813	86%
6	37411	18396	45034	22502	5776	85%
7	3921	14260	1962	17447	3967[c]	50%
8	48678	15732	37340	32866	9291	81%
9	13797[c]	12806	43695	2267[c]	58973	84%
10	13067	31268	10515	38569	12294	20%
11	13971[c]	6847[c]	42234	2959[c]	54515	79%
12	17390	93475	10866[c]	97277	1492	91%
Média	24501	37257	21800	37058	16047	70%

Nas tabelas 7.12 e 7.13 podemos visualizar a proporção do desempenho entre os SGBD obtida por meio da média dos tempos de todas as consultas e das consultas sobre a coleção que mais as favoreceram. A razão apresentada foi obtida por meio da divisão dos tempos médios das consultas nos SGBD da primeira linha sobre os tempos médios das consultas nos SGBD da primeira coluna.

A partir desses resultados é possível notar que o ElasticSearch foi superior a os outros SGBD. Por exemplo, observa-se que o ElasticSearch, nos tempos das execuções consecutivas das consultas sobre as coleções que mais as favoreceram (3a. linha da Tabela 7.13), ganhou de todos, sendo seu ganho mais expressivo sobre o OrientDB, onde a razão das

[c]Consulta cujo melhor tempo foi obtido utilizando índice composto
[i]Consulta onde foi possível utilizar índices

TAB. 7.8: Tempos das primeiras execuções das consultas do TPC-DS nas diferentes coleções no ES em milissegundos.

Consulta	Store_Sales	Customer	Date	Item	Time	Ganho
3	4236	5232	3027	1683	5360	60%
6	4623	3625	2749	3169	3848	41%
19	4837	4174	2729	2961	3925	44%
27	7151	6765	3663	5268	6712	49%
53	9808	12519	7961	4031	12167	59%
68	10503	9595	4739	8688	9318	55%
71	7295	7391	2233	4436	4960	69%
79	9613	9151	5057	8341	9018	47%
88	6050	4011	4669	4559	2814	53%
89	9122	8738	4279	5230	8575	53%
96	4547	3789	3335	3085	1193	74%
98	8029	10291	5504	3206	3584	60%
Média	7151	7107	4163	4555	5956	55%

TAB. 7.9: Tempos das primeiras execuções das consultas do TPC-DS nas diferentes coleções no OrientDB em milissegundos, utilizando índices.

Consulta	Customer	Date	Item	Time	Store_Sales	Ganho
3	209659	58068^{i}	3460^{i}	101309	87421	96%
6	291611	60409^{i}	14644^{i}	136424	64697	77%
19	10497^{i}	8784^{i}	2086^{i}	81833	73697	97%
27	47908^{i}	42773^{i}	295983	109793	69904^{i}	39%
53	225027	58165^{i}	93974^{i}	103096	85465	32%
68	155338^{i}	11981^{i}	96019	104990	182302^{i}	93%
71	198030	9774^{i}	11227^{i}	38568^{i}	73446	87%
79	95509^{i}	27414^{i}	106637	110358	78675	65%
88	48768^{i}	91353	93834	2410^{i}	30225^{i}	92%
89	249050	45543^{i}	28971^{i}	110231	83141	65%
96	48768^{i}	90701	84781	3758^{i}	35857^{i}	90%
98	215035	4678^{i}	31029^{i}	111558	83696	94%
Média	140342	41889	69600	84527	70397	77%

TAB. 7.10: Média dos tempos da primeira execução de todas as consultas e das consultas sobre a coleção com a modelagem que mais favoreceu cada consulta do TPC-DS.

SGBD	Média Total	Média Melhores
MongoDB	27333	5248
OrientDB	85506	17510
ElasticSearch	5786	3199

médias totais extraídas da Tabela 7.11 (47891/191) mostra que este foi 250 vezes mais rápido.

Apesar do MongoDB ter obtido uma média de tempo 1,64 vezes maior que o Elastic-

TAB. 7.11: Média dos tempos de execução de todas as consultas e das consultas sobre a coleção com a modelagem que mais favoreceu cada consulta do TPC-DS, realizadas consecutivamente.

SGBD	Média Total	Média Melhores
MongoDB	9528	1534
OrientDB	47891	8462
ElasticSearch	191	69

TAB. 7.12: Razão entre o desempenho dos SGBD nas primeiras execuções de todas as consultas e das primeiras execuções sobre as coleções que mais favoreceram cada consulta. T - Todas as consultas. M - Consultas sobre a coleção que mais favoreceu.

SGBD	Mongo		OrientDB		Elastic	
	T	M	T	M	T	M
MongoDB	-	-	3,13	3,34	0,21	0,61
OrientDB	0,32	0,3	-	-	0,07	0,18
ElasticSearch	4,72	1,64	14,78	5,47	-	-

TAB. 7.13: Razão entre o desempenho dos SGBD nas execuções consecutivas de todas as consultas e das execuções consecutivas sobre as coleções que mais favoreceram cada consulta. T - Todas as consultas. M - Consultas sobre a coleção que mais favoreceu.

SGBD	Mongo		OrientDB		Elastic	
	T	M	T	M	T	M
MongoDB	-	-	5,03	5,52	0,02	0,05
OrientDB	0,2	0,18	-	-	0	0,01
ElasticSearch	49,86	22,19	250,62	122,38	-	-

Search para as consultas executadas pela primeira vez nas coleções que mais favoreciam as consultas, em apenas uma dessas consultas o ElasticSearch foi mais rápido que o MongoDB, podendo considerar o MongoDB mais rápido que o ElasticSearch para consultas executadas pela primeira vez na coleção que mais favorece a consulta.

O OrientDB na média das consultas foi inferior a todos os SGBD tanto na média de todas as consultas quanto na média das consultas sobre as coleções que mais favoreciam as consultas.

O melhor desempenho do ElasticSearch está relacionado ao fato de que este cria índices invertidos (GARCIA-MOLINA et al., 2000) para todos os campos já na inserção dos documentos[22], permitindo um bom desempenho nas primeiras consultas, e também devido à ausência da necessidade de realizar a operação *unwind*.

Nas consultas realizadas consecutivamente o motivo do ElasticSearch ter obtido um desempenho muito superior aos outros SGBD se deve ao fato deste criar os chamados

[22]https://www.elastic.co/guide/en/elasticsearch/guide/current/analysis-intro.html

bit-sets explicados na seção 2.2.2.

O MongoDB por sua vez utiliza uma estrutura do tipo árvore B para seus índices[23], onde os valores são ordenados e é utilizado um algoritmo de busca binária para encontrar os documentos que combinam com determinado filtro. A razão do tempo entre a primeira consulta e as subsequentes (na ordem de 2,9 vezes maior), evidenciada pelas tabelas 7.10 e 7.11, está relacionada ao tempo de carregamento dos índices na memória.

Entre os SGBD estudados, o MongoDB é o que possui a melhor posição no ranking do site DB-Engines[24]. Este site, para fazer o seu ranking, se baseia no número de citações sobre os sistemas em websites, no interesse geral pelo sistema (baseado no número de consultas sobre o sistema no google), entre outros parâmetros. Entretanto, a partir dos resultados encontrados nas comparações de desempenho, este SGBD foi superado pelo ElasticSearch.

O OrientDB entre os SGBD estudados foi o que apresentou o pior desempenho nas consultas quando utilizando índices, onde o seu baixo desempenho pode estar relacionado com o fato de somente ser possível utilizar índices em campos das raízes dos documentos, pois assim como no MongoDB os índices criados utilizam uma estrutura de árvore B.

7.2.1 CONCLUSÕES DAS COMPARAÇÕES DE DESEMPENHO

Ao comparar o desempenho do MongoDB e do OrientDB sem utilizar índices pode-se concluir que de maneira geral, o MongoDB foi superior e que em apenas algumas consultas, principalmente nas executadas pela primeira vez, o OrientDB utilizando documentos ligados foi mais rápido. Ao utilizar índices foram comparados três SGBD, onde o ElasticSearch obteve um melhor desempenho na maioria das consultas, seguido pelo MongoDB. Por fim, o OrientDB foi o SGBD com o pior desempenho.

[23] http://jordankobellarz.github.io/mongodb/2015/07/31/mongodb-otimizando-com-indices.html
[24] http://db-engines.com/en/ranking

8 CONCLUSÕES

Os SGBD NoSQL são sistemas que foram criados a partir da necessidade de se gerenciar um volume maior de dados e aumentar o desempenho dos aplicativos e dos serviços de internet à medida em que este volume de dados aumenta. Entretanto, devido ao fato desses sistemas serem recentes, pouco se conhece a respeito da modelagem de dados neste tipo de SGBD e como a modelagem pode ser utilizada de forma a otimizar estes sistemas.

Apesar deste tipo de SGBD ter sido criado em busca de uma melhor performance em operações OLTP, dentre os modelos de dados NoSQL existentes, a maioria das implementações do modelo orientado a documento também permite que consultas analíticas possam ser realizadas de forma similar às consultas realizadas pelos SGBD relacionais, de forma que este modelo de SGBD também possa ser utilizado em operações OLAP. Entretanto a modelagem de dados para este tipo de aplicação pode ser diferente da modelagem para aplicações que utilizam os bancos de dados em operações OLTP.

Embora existam alguns trabalhos (HOBERMAN, 2014) (FOWLER; SADALAGE, 2013) (KAUR; RANI, 2013) que buscam orientar como os dados devem ser modelados de forma a otimizar as operações OLTP nesse tipo de SGBD, e trabalhos que avaliem o desempenho desses sistemas neste tipo de operação (BRITO, 2011) (PHAN et al., 2014) (ABRAMOVA et al., 2014) (ABUBAKAR et al., 2014), poucos trabalhos orientam como os dados podem ser modelados de forma a serem utilizados em operações analíticas (CHEVALIER et al., 2015) ou medem a capacidade desses sistemas na realização de consultas analíticas (CARNIEL et al., 2012b). Além disso, não se encontram trabalhos que propõem formas diferentes de modelar os dados para atender consultas analíticas e que demostrem como a modelagem pode influenciar neste tipo de aplicação.

Dentro deste contexto, este trabalho buscou preencher este vazio e investigar formas diferentes de modelar os dados nos SGBD orientados a documentos e como estas variações podem influenciar no desempenho desses SGBD, em consultas analíticas.

A partir da análise do impacto da modelagem dos agregados em cada SGBD foi possível confirmar a hipótese levantada, pois cada implementação de SGBD teve um comportamento diferente à medida em que diferentes modelagens eram utilizadas, e devido a este fator, este trabalho propõe heurísticas de como a modelagem de dados pode ser utilizada de forma a obter uma melhor performance nas implementações estudadas.

Para o estudo do impacto da variação da modelagem de agregados em consultas

analíticas foi utilizado um *benchmark* conhecido e utilizado para avaliar SGBD relacionais no apoio a aplicações voltadas a sistemas de apoio à decisão (aplicações analíticas). A partir dos dados desse *benchmark* foi possível criar agregados (documentos) utilizando diferentes formas de modelagem. Cada modelagem permitiu a geração de uma diferente coleção de documentos.

A partir da realização de consultas analíticas nos agregados em diferentes modelagens nas diferentes implementações de SGBD, foi possível observar e identificar diferentes influências no desempenho dessas consultas. Ao final do estudo do comportamento de cada SGBD foi possível propor, para alguns dos SGBD estudados, heurísticas que podem ser utilizadas para incrementar a performance dos SGBD nesse tipo de consulta.

As heurísticas criadas consistem em passos a serem seguidos de forma a obter um menor conjunto de dados replicados em coleções que utilizam diferentes modelagens de agregados, que permitem um aumento significativo no desempenho dos SGBD orientados a documento em consultas analíticas.

A partir das heurísticas propostas, estas foram aplicadas em outro conjunto de dados oriundo de outro *benchmark*, o TPC-DS, que possui objetivos similares ao utilizado para o estudo inicial, permitindo que as heurísticas fossem testadas e validadas.

Como complemento ao estudo principal deste trabalho foi realizada uma análise do impacto da variação da modelagem dos agregados nos SGBD estudados ao utilizar o recurso de índices, onde foram observados comportamentos diferentes dos observados quando este recurso não era utilizado.

Após a análise do comportamento dos SGBD foi realizada uma comparação de desempenho entre os SGBD estudados, onde foi observado que apesar de haver uma superioridade generalizada de alguns SGBDs, dependendo da forma com que os dados forem modelados e a forma com que as consultas forem realizadas, esta situação pode ser invertida. Uma implementação que teve um menor desempenho em consultas consecutivas pode ser mais rápida nas consultas realizadas em intervalos maiores, ou quando as consultas são realizadas em modelagens de agregados que as favorecem.

8.1 CONTRIBUIÇÕES

As principais contribuições realizadas por esta dissertação são as seguintes:

- Nova forma de se pensar na modelagem de dados para os SGBD NoSQL orientados a documentos, voltada para consultas OLAP.

- Estudo do impacto da variação da modelagem dos agregados em cinco diferentes SGBD orientados a documento, utilizando e não utilizando índices.
- Proposição de heurísticas de modelagens de agregados, que possibilitam a melhoria do desempenho de 3 implementações de SGBD, sem a utilização de índices.
- Estudo da capacidade de utilizar índices em estruturas de dados aninhados em quatro SGBD e do aumento do desempenho em consultas analíticas ao utilizar este recurso.
- Comparação de desempenho em consultas analíticas entre dois SGBD orientados a documentos, sem a utilização de índices.
- Comparação de desempenho em consultas analíticas entre quatro SGBD orientados a documentos, com a utilização de índices.
- Adaptação de dois conhecidos *benchmarks* voltados para sistemas de apoio à decisão para sua aplicação em SGBD orientados a documentos.
- Disponibilização em domínio público da adaptação das consultas de dois *benchmarks* conhecidos da linguagem SQL para a linguagem de consulta de cinco SGBD orientados a documentos.
- Disponibilização em domínio público de consultas na linguagem SQL que possibilitam a transformação de dados modelados conforme um esquema relacional para o formato JSON.

8.2 LIMITAÇÕES E TRABALHOS FUTUROS

Apesar do estudo ter demostrado o impacto da modelagem de agregados e ter permitido a criação de heurísticas de forma a incrementar o desempenho em uma parcela de SGBD NoSQL orientados a documentos existentes em consultas analíticas, algumas limitações foram encontradas no estudo proposto.

Grande parte dos estudos a respeito dos SGBD NoSQL, avaliam estes sistemas utilizando os SGBD em um ambiente distribuído. Entretanto, em virtude deste trabalho ter como foco a análise do impacto da modelagem no desempenho, a distribuição dos dados poderia causar uma análise errônea do desempenho em virtude da latência de rede, o que poderia levar a conclusões erradas a respeito do comportamento de cada SGBD à medida em que a modelagem do agregado era alterada. Além disso, em virtude de não

ter sido encontrado na literatura outro trabalho que tivesse o mesmo foco deste trabalho, considerou-se que este trabalho traria uma boa contribuição. Assim, foi decidido realizar os experimentos em um ambiente não distribuído, e a análise do impacto da modelagem em ambientes distribuídos ficou atribuída a trabalhos futuros.

Além disso, em virtude da possibilidade dos SGBD NoSQL em distribuir os dados e com isso utilizar em cada nó uma quantidade de dados que seja compatível com a memória principal do sistema, a quantidade de dados que se buscou ser utilizada neste trabalho foi proporcional à memória principal da máquina utilizada, visando simular o comportamento dentro de um nó. Portanto, não é possível afirmar que as heurísticas se manterão válidas para conjuntos de dados maiores que a capacidade da memória principal.

O presente trabalho utilizou as tabelas oriundas de dois esquemas estrela como base para as diferentes modelagens propostas, em virtude deste tipo de esquema ser muito utilizado em Data Warehouses e pela necessidade de limitar o trabalho para um único modelo de esquema conceitual. Portanto, as heurísticas apresentadas se limitam a orientar a escolha de modelagens de agregados cuja modelagem conceitual possuem como padrão este modelo de esquema ou um modelo que possa ser adaptado para este, de forma semelhante ao realizado no esquema Store_Sales do TPC-DS neste trabalho.

Outra limitação, porém de pouca importância, encontrada neste trabalho foi a necessidade de excluir das consultas a operação de ordenação, principalmente devido à forma que o ES realiza a operação de agrupamento e ordenação dos campos da consulta, desta forma, o tempo de execução das consultas nos demais SGBD não foi prejudicada devido à utilização da cláusula de ordenação.

Apesar de ter sido possível discutir o comportamento do MarkLogic, a restrição com relação à apresentação dos tempos de execução das suas consultas pode ser considerada uma limitação deste trabalho, pois não é possível para o leitor acompanhar o comportamento deste SGBD por meio da leitura dos tempos das consultas e devido à necessidade de excluir este SGBD das comparações de desempenho.

Em virtude deste trabalho ter apresentado uma nova forma de se pensar na modelagem dos dados em SGBD orientados a documentos, voltada para consultas analíticas, inúmeros trabalhos futuros podem ser realizados:

- Estudo do impacto da modelagem de dados e mais especificamente de agregados em outras implementações de SGBD orientados a documentos.

- Estudo do impacto da modelagem de dados e mais especificamente de agregados em SGBD NoSQL orientado a documentos com uma quantidade maior de dados.

- Estudo do impacto da modelagem de dados e mais especificamente de agregados em SGBD NoSQL orientado a documentos em ambiente distribuído.
- Estudo do impacto da modelagem em dados de tipos diferentes dos utilizados neste trabalho ou em estruturas com dados aninhados em uma quantidade maior de níveis.
- Estudo do impacto da modelagem de dados em operações OLTP, buscando obter o custo de manutenção de mais de uma modelagem de agregados.
- Criação de heurísticas para escolha de modelagens utilizando o recurso de índices.

9 REFERÊNCIAS BIBLIOGRÁFICAS

ABRAMOVA, V.; BERNARDINO, J. ; FURTADO, P. Experimental evaluation of nosql databases. **International Journal of Database Management Systems**, v. 6, n. 3, p. 1, 2014.

ABUBAKAR, Y.; ADEYI, T. S. ; AUTA, I. G. Performance evaluation of nosql systems using ycsb in a resource austere environment. **IJAIS**, v. 7, n. 8, p. 23–27, 2014.

BRASETVIK, A. Elasticsearch as a NoSQL Database. Disponível em: <https://www.elastic.co/blog/found-elasticsearch-as-nosql>. Acesso em: 31 mar. de 2016.

RICARDO W. BRITO. Bancos de dados NoSQL x SGBDs relacionais: Analise Comparativa. Disponível em: <http://www.infobrasil.inf.br/userfiles/27-05-S4-1-68840-Bancos de Dados NoSQL.pdf>. Acesso em: 6 nov. de 2015.

BUGIOTTI, F.; CABIBBO, L.; ATZENI, P. ; TORLONE, R. Database design for nosql systems. **International Conference on Conceptual Modeling**, v. 8824, p. 223–231, 2014.

CARNIEL, A. C.; DE AGUIAR SA, A.; RIBEIRO, M. X.; BUENO, R.; CIFERRI, C. D. A. ; CIFERRI, R. Análise experimental de bases de dados relacionais e nosql no processamento de consultas sobre data warehouses. **Simpósio Brasileiro de Banco de Dados**, v. 1, p. 113–120, 2012.

CARNIEL, A. C.; DE AGUIAR SÁ, A.; BRISIGHELLO, V. H. P.; RIBEIRO, M. X.; BUENO, R.; CIFERRI, R. R. ; DE AGUIAR CIFERRI, C. D. Query processing over data warehouse using relational databases and nosql. In: SIMPOSIO BRASILEIRO DE BANCO DE DADOS, 27., 2012, São Paulo. **Anais**... [S.l.: s.n.], 2012, p. 113–120.

CHAN, C.-Y.; IOANNIDIS, Y. E. Bitmap index design and evaluation. In: ACM SIGMOD RECORD, 24., 1998. **Anais**... [S.l.: s.n.], 1998, p. 355–366.

CHEVALIER, M.; EL MALKI, M.; KOPLIKU, A.; TESTE, O. ; TOURNIER, R. Implementing multidimensional data warehouses into nosql. In: 17TH INTERNATIONAL CONFERENCE ON ENTERPRISE INFORMATION SYSTEMS (ICEIS'15), SPAIN, 17., 2015. **Anais**... [S.l.: s.n.], 2015, p. 172–183.

CHODOROW, K. Mongodb: The definitive guide. In: SPENCER, A. (Org.). **MongoDB: The Definitive Guide**. 2. ed. Sebastopol: O'Reilly Media, 2013. p. 140.

ELMASRI, R.; LI, Q.; FU, J.; WU, Y.-C.; HOJABRI, B. ; ANDE, S. Conceptual modeling for customized xml schemas. **Data & Knowledge Engineering**, v. 54, n. 1, p. 57–76, 2005.

EVANS, E. **Domain-driven design: tackling complexity in the heart of software**. [S.l.]: Addison-Wesley Professional, 2004.

FONG, J. Mapping extended entity relationship model to object modeling technique. **ACM Sigmod Record**, v. 24, n. 3, p. 18–22, 1995.

THE APACHE SOFTWARE FOUNDATION. CouchDB Relax. Disponível em: <http://docs.couchdb.org/en/1.6.1/intro/index.html>. Acesso em: 20 abr. de 2016.

FOWLER, M.; SADALAGE, P. J. Nosql um guia consiso para o mundo emergente da persistência poliglota. In: PRATES, R. (Org.). **NoSQL Um Guia Consiso para o mundo Emergente da Persistência Poliglota**. São Paulo: Novatec Editora Ltda, 2013. p. 133.

GANDINI, A.; GRIBAUDO, M.; KNOTTENBELT, W. J.; OSMAN, R. ; PIAZZOLLA, P. Performance evaluation of nosql databases. In: HORVÁTH, A.; WOLTER, K. (Org.). **Computer Performance Engineering: 11th European Workshop, EPEW 2014, Florence, Italy, September 11-12, 2014. Proceedings**. Cham: Springer International Publishing, 2014. p. 16–29. ISBN 978-3-319-10885-8.

GARCIA-MOLINA, H.; ULLMAN, J. D. ; WIDOM, J. **Database system implementation**. [S.l.]: Prentice Hall Upper Saddle River, NJ:, 2000.

GARULLI, L.; OROBETS, A.; LOMAKIN, A.; DELL'AQUILA, L.; MOLINO, L. OrientDB Manual - version 2.1.x. Disponível em: <http://orientdb.com/docs>. Acesso em: 31 mar. de 2016.

GORMLEY, C.; TONG, Z. Optimistic concurrency control. In: LOUKIDES, M.; ANDERSON, B. (Org.). **Elasticsearch: The Definitive Guide**. Sebastopol: O'Reilly, 2015. p. 47.

HEUSER, C. A. **Projeto de banco de dados: Volume 4 da Série Livros didáticos informática UFRGS**. 4. ed. Porto Alegre: Bookman Editora, 2009.

HOBERMAN, S. **Data Modeling for MongoDB: Building Well-Designed and Supportable MongoDB Databases**. [S.l.]: Technics Publications, 2014.

HOWS, D.; MEMBREY, P.; PLUGGE, E. ; HAWKINS, T. The definitive guide to mongodb. In: LOWMAN, M. (Org.). **Complete guide to dealing with Big Data using MongoDB**. 3. ed. Nova Iorque: Apress, 2015. p. 251.

HUNTER, J.; WOOLDRIDGE, M. **Inside MarkLogic Server**. San Carlos: MarkLogic Corporation, 2016.

KAUR, K.; RANI, R. Modeling and querying data in nosql databases. In: BIG DATA, 2013 IEEE INTERNATIONAL CONFERENCE ON, 2., 2013. **Anais**... [S.l.: s.n.], 2013, p. 1–7.

KOCH, R. **O princípio 80/20: o segredo de se realizar mais com menos**. Rio de Janeiro: Rocco, 2000.

KONONENKO, O.; BAYSAL, O.; HOLMES, R. ; GODFREY, M. W. Mining modern repositories with elasticsearch. In: PROCEEDINGS OF THE 11TH WORKING CONFERENCE ON MINING SOFTWARE REPOSITORIES, 11., 2014. **Anais**... [S.l.: s.n.], 2014, p. 328–331.

NAVATHE, E. Sistemas de bancos de dados. In: PEARSON (Org.). **Sistemas de bancos de dados**. Berlim: PEARSON EDUCATION, 2014. p. 492–493.

NAVATHE, E. Sistemas de bancos de dados. In: PEARSON (Org.). **Sistemas de bancos de dados**. Berlim: PEARSON EDUCATION, 2014. p. 463.

O'NEIL, P.; O'NEIL, B.; CHEN, X. ; REVILAK, S. The star schema benchmark and augmented fact table indexing. **TPCTC**, v. 5895, p. 237–252, 2009.

PHAN, T. A. M.; NURMINEN, J. K. ; DI FRANCESCO, M. Cloud databases for internet-of-things data. In: IEEE INTERNATIONAL CONFERENCE ON INTERNET OF THINGS, 2014., 2014. **Anais**... [S.l.: s.n.], 2014, p. 117–124.

SCABORA, L. C.; BRITO, J. J.; CIFERRI, R. R.; CIFERRI, C. D. D. A. ; OTHERS. Physical data warehouse design on nosql databases olap query processing over hbase. **International Conference on Enterprise Information Systems, XVIII**, v. 1, p. 111–118, 2016.

SCHROEDER, R.; MELLO, R. D. S. Improving query performance on xml documents: a workload-driven design approach. In: PROCEEDINGS OF THE EIGHTH ACM SYMPOSIUM ON DOCUMENT ENGINEERING, 8., 2008. **Anais**... [S.l.: s.n.], 2008, p. 177–186.

10 APÊNDICES

```
SELECT SUM(lo_extendedprice*lo_discount) as revenue
FROM lineorder, date
WHERE lo_orderdate = d_datekey
AND d_year = 1993
AND lo_discount between1 AND 3
AND lo_quantity < 25;
```

FIG. 10.1: Consulta 1.1 do SSB expressa na linguagem SQL.

```
SELECT SUM(lo_extendedprice*lo_discount) as revenue
FROM lineorder, date
WHERE lo_orderdate = d_datekey
AND d_yearmonthnum = 199401
AND lo_discount between4 AND 6
AND lo_quantity between 26 AND 35;
```

FIG. 10.2: Consulta 1.2 do SSB expressa na linguagem SQL.

```
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;
```

FIG. 10.3: Consulta 1.3 do SSB expressa na linguagem SQL.

```
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 ;
```

FIG. 10.4: Consulta 2.1 do SSB expressa na linguagem SQL.

```
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 ;
```

FIG. 10.5: Consulta 2.2 do SSB expressa na linguagem SQL.

```
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 ;
```

FIG. 10.6: Consulta 2.3 do SSB expressa na linguagem SQL.

```
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 ;
```

FIG. 10.7: Consulta 3.1 do SSB expressa na linguagem SQL.

```
SELECT c_city, s_city, d_year, SUM(lo_revenue) as reve-
nue 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 ;
```

FIG. 10.8: Consulta 3.2 do SSB expressa na linguagem SQL.

```
SELECT c_city, s_city, d_year, SUM(lo_revenue) as reve-
nue 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;
```

FIG. 10.9: Consulta 3.3 do SSB expressa na linguagem SQL.

```
SELECT c_city, s_city, d_year, SUM(lo_revenue) as reve-
nue 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;
```

FIG. 10.10: Consulta 3.4 do SSB expressa na linguagem SQL.

```
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 ;
```

FIG. 10.11: Consulta 4.1 do SSB expressa na linguagem SQL.

```
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;
```

FIG. 10.12: Consulta 4.2 do SSB expressa na linguagem SQL.

```
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 p_category = 'MFGR#14'
GROUP BY d_year, s_city, p_brand1;
```

FIG. 10.13: Consulta 4.3 do SSB expressa na linguagem SQL.

```
define AGGC= text({"ss_ext_sales_price",1},{"ss_sales_price",1},
{"ss_ext_discount_amt",1},{"ss_net_profit",1});
define MONTH = random(11,12,uniform);
define MANUFACT= random(1,1000,uniform);
define _LIMIT=100;

[_LIMITA] select [_LIMITB] dt.d_year,item.i_brand_id brand_id,
item.i_brand brand, sum([AGGC]) sum_agg
from  date_dim dt , store_sales ,item
where dt.d_date_sk = store_sales.ss_sold_date_sk
and store_sales.ss_item_sk = item.i_item_sk
and item.i_manufact_id = [MANUFACT]
and dt.d_moy=[MONTH]
group by dt.d_year,item.i_brand,item.i_brand_id
order by dt.d_year,sum_agg desc,brand_id
[_LIMITC];
```

FIG. 10.14: Consulta 3 original do TPC-DS expressa na linguagem SQL.

```
select  dt.d_year ,item.i_brand_id brand_id ,item.i_brand brand ,
sum(ss_sales_price)
from  date_dim dt ,store_sales ,item
where dt.d_date_sk = store_sales.ss_sold_date_sk
and store_sales.ss_item_sk = item.i_item_sk
and item.i_manufact_id = 600 and dt.d_moy=12
group by dt.d_year,item.i_brand ,item.i_brand_id;
```

FIG. 10.15: Consulta 3 alterada do TPC-DS expressa na linguagem SQL.

```
define YEAR = random(1998, 2002, uniform);
define MONTH= random(1,7,uniform);
define _LIMIT=100;

[_LIMITA] select [_LIMITB] a.ca_state state, count(*) cnt
from customer_address a,customer c,store_sales s,date_dim d, tem i
where a.ca_address_sk = c.c_current_addr_sk
and c.c_customer_sk = s.ss_customer_sk
and s.ss_sold_date_sk = d.d_date_skand s.ss_item_sk = i.i_item_sk
and d.d_month_seq =
(select distinct (d_month_seq)
from date_dim
where d_year = [YEAR] and d_moy = [MONTH] )
and i.i_current_price > 1.2 *
(select avg(j.i_current_price)
from item j where j.i_category = i.i_category)
group by a.ca_state
having count(*) >= 10
order by cnt
[_LIMITC];
```

FIG. 10.16: Consulta 6 original do TPC-DS expressa na linguagem SQL.

```
select c.ca_state state, count(*) cnt
from customer1 c ,store_sales s ,date_dim d ,item i
where c.c_customer_sk = s.ss_customer_sk
and s.ss_sold_date_sk = d.d_date_sk
and s.ss_item_sk = i.i_item_sk
and d_year = 2001
and i.i_current_price > 9
group by c.ca_state;
```

FIG. 10.17: Consulta 6 alterada do TPC-DS expressa na linguagem SQL.

```
define YEAR= random(1998, 2002, uniform);
define MONTH=random(11,12,uniform);
define MGR_IDX = dist(i_manager_id, 1, 1);
define MANAGER=random(distmember(i_manager_id, [MGR_IDX], 2),
distmember(i_manager_id, [MGR_IDX], 3),uniform);
define _LIMIT=100;

[_LIMITA]  select [_LIMITB] i_brand_id brand_id, i_brand brand,
i_manufact_id, i_manufact,
sum(ss_ext_sales_price) ext_price
from date_dim, store_sales, item,customer,customer_address,store
where d_date_sk = ss_sold_date_skand ss_item_sk = i_item_sk
and i_manager_id=[MANAGER]and d_moy=[MONTH]
and d_year=[YEAR] and ss_customer_sk = c_customer_sk
and c_current_addr_sk = ca_address_sk
and substr(ca_zip,1,5) <> substr(s_zip,1,5)
and ss_store_sk = s_store_sk
group by i_brand,i_brand_id ,i_manufact_id ,i_manufact
order by ext_price desc, i_brand, i_brand_id, i_manufact_id, i_manufact
[_LIMITC] ;
```

FIG. 10.18: Consulta 19 original do TPC-DS expressa na linguagem SQL.

```
select  i_brand_id brand_id, i_brand brand, i_manufact_id, i_manufact,
sum(ss_ext_sales_price) ext_price
from date_dim, store_sales, item, customer1
where d_date_sk = ss_sold_date_sk
and ss_item_sk = i_item_sk
and ss_customer_sk = c_customer_sk
and i_manager_id=20
and d_moy=11 and d_year=2001
and ca_zip ='35709'
group by i_brand ,i_brand_id ,i_manufact_id ,i_manufact;
```

FIG. 10.19: Consulta 19 alterada do TPC-DS expressa na linguagem SQL.

```
define YEAR = random(1998, 2002, uniform);
define GEN= dist(gender, 1, 1);
define MS= dist(marital_status, 1, 1);
define ES= dist(education, 1, 1);
define STATENUMBER=ulist(random(1, rowcount("active_states",
"store"), uniform),6);
define STATE_A=distmember(fips_county,[STATENUMBER.1], 3);
define STATE_B=distmember(fips_county,[STATENUMBER.2], 3);
define STATE_C=distmember(fips_county,[STATENUMBER.3], 3);
define STATE_D=distmember(fips_county,[STATENUMBER.4], 3);
define STATE_E=distmember(fips_county,[STATENUMBER.5], 3);
define STATE_F=distmember(fips_county,[STATENUMBER.6], 3);
define _LIMIT=100;

 [_LIMITA] select [_LIMITB] i_item_id,
 s_state, grouping(s_state) g_state, avg(ss_quantity) agg1,
avg(ss_list_price) agg2, avg(ss_coupon_amt) agg3, avg(ss_sales_price)
agg4
from store_sales, customer_demographics, date_dim, store, item
where ss_sold_date_sk = d_date_skand
ss_item_sk = i_item_skand ss_store_sk = s_store_skand
ss_cdemo_sk = cd_demo_skand cd_gender = '[GEN]'and
cd_marital_status = '[MS]'and cd_education_status = '[ES]'and
d_year = [YEAR]and s_state in ('[STATE_A]','[STATE_B]', '[STATE_C]',
'[STATE_D]', '[STATE_E]', '[STATE_F]')
group by rollup (i_item_id, s_state)
order by i_item_id, s_state
[_LIMITC];
```

FIG. 10.20: Consulta 27 original do TPC-DS expressa na linguagem SQL.

```
select i_item_id, s_state, sum(ss_sales_price) agg4
from store_sales, customer1, date_dim, item
where ss_sold_date_sk = d_date_skand
ss_item_sk = i_item_skand
ss_customer_sk = c_customer_skand
cd_gender = 'F'and
cd_marital_status = 'S'and
cd_education_status = 'Secondary'and
d_year = 2001and
s_state in ('TN')
group by i_item_id, s_state;
limit 100;
```

FIG. 10.21: Consulta 27 alterada do TPC-DS expressa na linguagem SQL.

```
define DMS = random(1176,1224,uniform);
define _LIMIT=100;

[_LIMITA] select [_LIMITB] * from
(select i_manufact_id,
sum(ss_sales_price) sum_sales,
avg(sum(ss_sales_price)) over (partition by i_manufact_id)
avg_quarterly_sales
from item, store_sales, date_dim, store
where ss_item_sk = i_item_skand
ss_sold_date_sk = d_date_skand
ss_store_sk = s_store_skand
d_month_seq in ([DMS],[DMS]+1,[DMS]+2,[DMS]+3,[DMS]+4,[DMS]
+5,[DMS]+6,[DMS]+7,[DMS]+8,[DMS]+9,[DMS]+10,[DMS]+11)and
((i_category in ('Books','Children','Electronics')and
i_class in ('personal','portable','reference','self-help')and
i_brand in ('scholaramalgamalg #14','scholaramalgamalg #7',
'exportiunivamalg #9','scholaramalgamalg #9'))
or(i_category in ('Women','Music','Men')and
i_class in ('accessories','classical','fragrances','pants')and
i_brand in ('amalgimporto #1','edu packscholar #1','exportiimporto #1',
'importoamalg #1')))
group by i_manufact_id, d_qoy ) tmp1
where case when avg_quarterly_sales > 0
then abs (sum_sales - avg_quarterly_sales)/ avg_quarterly_sales
else null end > 0.1
order by avg_quarterly_sales,
 sum_sales,
 i_manufact_id
[_LIMITC];
```

FIG. 10.22: Consulta 53 original do TPC-DS expressa na linguagem SQL.

```
select i_item_id , i_item_desc, i_category, i_class, i_current_price ,
sum(ss_ext_sales_price) as itemrevenue
from store_sales, item, date_dim
where ss_item_sk = i_item_sk
and ss_sold_date_sk = d_date_sk
and i_category in ('Music','Home','Sports')
and d_month_seq between 1200and 1211
group by i_item_id , i_item_desc, i_category , i_class , i_current_price;
```

FIG. 10.23: Consulta 53 alterada do TPC-DS expressa na linguagem SQL.

```
define DEPCNT=random(0,9,uniform);
define YEAR = random(1998,2000,uniform);
define VEHCNT=random(-1,4,uniform);
define CITYNUMBER = ulist(random(1, rowcount("active_cities",
"store"), uniform), 2);
define CITY_A = distmember(cities, [CITYNUMBER.1], 1);
define CITY_B = distmember(cities, [CITYNUMBER.2], 1);
define _LIMIT=100;

[_LIMITA] select [_LIMITB] c_last_name ,c_first_name, ca_city,
bought_city, ss_ticket_number,extended_price, extended_tax, list_price
 from (select ss_ticket_number, ss_customer_sk, ca_city bought_city,
sum(ss_ext_sales_price) extended_price, sum(ss_ext_list_price) list_price,
sum(ss_ext_tax) extended_tax
 from store_sales, date_dim, store, household_demographics,
customer_address
where store_sales.ss_sold_date_sk = date_dim.d_date_sk
and store_sales.ss_store_sk = store.s_store_sk
and store_sales.ss_hdemo_sk = household_demographics.hd_demo_sk
and store_sales.ss_addr_sk = customer_address.ca_address_sk
and date_dim.d_dom between 1and 2
and (household_demographics.hd_dep_count = [DEPCNT] or
 household_demographics.hd_vehicle_count= [VEHCNT])
and date_dim.d_year in ([YEAR],[YEAR]+1,[YEAR]+2)
and store.s_city in ('[CITY_A]','[CITY_B]')
 group by ss_ticket_number, ss_customer_sk, ss_addr_sk,ca_city) dn,
customer, customer_address current_addr
where ss_customer_sk = c_customer_sk
and customer.c_current_addr_sk = current_addr.ca_address_sk
and current_addr.ca_city <> bought_city
order by c_last_name, ss_ticket_number
[_LIMITC];
```

FIG. 10.24: Consulta 68 original do TPC-DS expressa na linguagem SQL.

```
select c_last_name ,c_first_name ,ca_city,ss_ticket_number ,
sum(ss_ext_sales_price) extended_price
from store_sales ,date_dim ,customer1
where store_sales.ss_sold_date_sk = date_dim.d_date_sk
and store_sales.ss_customer_sk = customer1.c_customer_sk
and date_dim.d_dom between 20and 21
and (customer1.hd_dep_count = 4 or
customer1.hd_vehicle_count= 2)
and date_dim.d_year in (1999,2000,2001)
and store_sales.s_city in ('Midway')
group by c_last_name ,c_first_name ,ca_city, ss_ticket_number;
```

FIG. 10.25: Consulta 68 alterada do TPC-DS expressa na linguagem SQL.

```
define YEAR= random(1998, 2002, uniform);
define MONTH=random(11,12,uniform);

 select i_brand_id brand_id, i_brand brand,t_hour,t_minute,
 sum(ext_price) ext_price
 from item, (select ws_ext_sales_price as ext_price,
ws_sold_date_sk as sold_date_sk, ws_item_sk as sold_item_sk,
ws_sold_time_sk as time_sk
from web_sales,date_dim
where d_date_sk = ws_sold_date_sk
and d_moy=[MONTH] and d_year=[YEAR]
union all
select cs_ext_sales_price as ext_price, cs_sold_date_sk as sold_date_sk,
cs_item_sk as sold_item_sk, cs_sold_time_sk as time_sk
from catalog_sales,date_dim
where d_date_sk = cs_sold_date_sk
and d_moy=[MONTH] and d_year=[YEAR]
union all
select ss_ext_sales_price as ext_price, ss_sold_date_sk as sold_date_sk,
ss_item_sk as sold_item_sk, ss_sold_time_sk as time_sk
from store_sales,date_dim
where d_date_sk = ss_sold_date_sk
and d_moy=[MONTH]
and d_year=[YEAR]) as tmp,time_dim
where sold_item_sk = i_item_sk
and i_manager_id=1
and time_sk = t_time_sk
and (t_meal_time = 'breakfast' or t_meal_time = 'dinner')
group by i_brand, i_brand_id,t_hour,t_minute
order by ext_price desc, i_brand_id ;
```

FIG. 10.26: Consulta 71 original do TPC-DS expressa na linguagem SQL.

```
select i_brand_id brand_id, i_brand brand,t_hour,t_minute,
sum(ss_ext_sales_price) ext_price
from item, date_dim, store_sales, time_dim1
where   ss_item_sk = i_item_sk
and ss_sold_time_sk = t_time_sk
and ss_sold_date_sk = d_date_sk
and i_manager_id=1
and d_moy=11
and d_year=2001
and (t_meal_time = 'breakfast' or t_meal_time = 'dinner')
group by i_brand, i_brand_id,t_hour,t_minute;
```

FIG. 10.27: Consulta 71 alterada do TPC-DS expressa na linguagem SQL.

```
define DEPCNT=random(0,9,uniform);
define YEAR = random(1998,2000,uniform);
define VEHCNT=random(-1,4,uniform);
define _LIMIT=100;

[_LIMITA] select [_LIMITB]
c_last_name,c_first_name,substr(s_city,1,30),ss_ticket_number,amt,profit
from  (select ss_ticket_number, ss_customer_sk, store.s_city,
sum(ss_coupon_amt) amt, sum(ss_net_profit) profit
from store_sales,date_dim,store,household_demographics
where store_sales.ss_sold_date_sk = date_dim.d_date_sk
and store_sales.ss_store_sk = store.s_store_sk
and store_sales.ss_hdemo_sk = household_demographics.hd_demo_sk
and (household_demographics.hd_dep_count = [DEPCNT] or
household_demographics.hd_vehicle_count > [VEHCNT])
and date_dim.d_dow = 1
and date_dim.d_year in ([YEAR],[YEAR]+1,[YEAR]+2)
and store.s_number_employees between 200and 295
group by ss_ticket_number,ss_customer_sk,ss_addr_sk,store.s_city)
ms,customer
where ss_customer_sk = c_customer_sk
order by c_last_name,c_first_name,substr(s_city,1,30), profit
[_LIMITC];
```

FIG. 10.28: Consulta 79 original do TPC-DS expressa na linguagem SQL.

```
select ss_ticket_number ,ss_customer_sk ,store_sales.s_city ,
sum(ss_coupon_amt) amt
from store_sales,date_dim,customer1
where store_sales.ss_sold_date_sk = date_dim.d_date_sk
and store_sales.ss_customer_sk = customer1.c_customer_sk
and (customer1.hd_dep_count = 5 or customer1.hd_vehicle_count > 2)
and date_dim.d_dow = 1
and date_dim.d_year in (1998,1999,2000)and
store_sales.s_number_employees between 200and 295
group by ss_ticket_number,ss_customer_sk,store_sales.s_city;
```

FIG. 10.29: Consulta 79 alterada do TPC-DS expressa na linguagem SQL.

```
define HOUR = ulist(random(-1,4,uniform),3);
define STORE = dist(stores,1,1);

select  *
from (select count(*) h8_30_to_9
from store_sales, household_demographics , time_dim, store
where ss_sold_time_sk = time_dim.t_time_sk
and ss_hdemo_sk = household_demographics.hd_demo_sk
and ss_store_sk = s_store_sk
and time_dim.t_hour = 8
and time_dim.t_minute >= 30
and ((household_demographics.hd_dep_count = [HOUR.1]and
household_demographics.hd_vehicle_count<=[HOUR.1]+2) or
(household_demographics.hd_dep_count = [HOUR.2]and
household_demographics.hd_vehicle_count<=[HOUR.2]+2) or
(household_demographics.hd_dep_count = [HOUR.3]and
household_demographics.hd_vehicle_count<=[HOUR.3]+2))
and store.s_store_name = 'ese') s1,
(select count(*) h9_to_9_30
from store_sales, household_demographics , time_dim, store
where ss_sold_time_sk = time_dim.t_time_sk
and ss_hdemo_sk = household_demographics.hd_demo_sk
and ss_store_sk = s_store_sk
and time_dim.t_hour = 9
and time_dim.t_minute < 30
and ((household_demographics.hd_dep_count = [HOUR.1]and
household_demographics.hd_vehicle_count<=[HOUR.1]+2) or
(household_demographics.hd_dep_count = [HOUR.2]and
household_demographics.hd_vehicle_count<=[HOUR.2]+2) or
(household_demographics.hd_dep_count = [HOUR.3]and
household_demographics.hd_vehicle_count<=[HOUR.3]+2))
and store.s_store_name = 'ese') s2,
```

FIG. 10.30: Consulta 88 original do TPC-DS expressa na linguagem SQL.

```
select count(*) h8_30_to_9
from store_sales, customer1 , time_dim1
where ss_sold_time_sk = time_dim1.t_time_sk
and ss_customer_sk = customer1.c_customer_sk
and time_dim1.t_hour = 8
and time_dim1.t_minute >= 30
and ((customer1.hd_dep_count = 1and customer1.hd_vehicle_count<=3)
or (customer1.hd_dep_count = 2and customer1.hd_vehicle_count<=4) or
(customer1.hd_dep_count = 3and customer1.hd_vehicle_count<=5))
and store_sales.s_store_name = 'ese';
```

FIG. 10.31: Consulta 88 alterada do TPC-DS expressa na linguagem SQL.

```
define YEAR = random(1998, 2002, uniform);
define IDX = ulist(random(1, rowcount("categories"), uniform), 6);
define CAT_A = distmember(categories, [IDX.1], 1);
define CLASS_A = DIST(distmember(categories, [IDX.1], 2), 1, 1);
define CAT_B = distmember(categories, [IDX.2], 1);
define CLASS_B = DIST(distmember(categories, [IDX.2], 2), 1, 1);
define CAT_C = distmember(categories, [IDX.3], 1);
define CLASS_C = DIST(distmember(categories, [IDX.3], 2), 1, 1);
define CAT_D = distmember(categories, [IDX.4], 1);
define CLASS_D = DIST(distmember(categories, [IDX.4], 2), 1, 1);
define CAT_E = distmember(categories, [IDX.5], 1);
define CLASS_E = DIST(distmember(categories, [IDX.5], 2), 1, 1);
define CAT_F = distmember(categories, [IDX.6], 1);
define CLASS_F = DIST(distmember(categories, [IDX.6], 2), 1, 1);
define _LIMIT=100;

[_LIMITA] select [_LIMITB] *
from(select i_category, i_class, i_brand, s_store_name, s_company_name,
d_moy, sum(ss_sales_price) sum_sales, avg(sum(ss_sales_price)) over
(partition by i_category, i_brand, s_store_name, s_company_name)
avg_monthly_sales from item, store_sales, date_dim, store
where ss_item_sk = i_item_sk and ss_sold_date_sk = d_date_sk and
ss_store_sk = s_store_sk and d_year in ([YEAR]) and
  ((i_category in ('[CAT_A]','[CAT_B]','[CAT_C]') and
i_class in ('[CLASS_A]','[CLASS_B]','[CLASS_C]')  )
or (i_category in ('[CAT_D]','[CAT_E]','[CAT_F]')and
i_class in ('[CLASS_D]','[CLASS_E]','[CLASS_F]')  ))
group by i_category, i_class, i_brand, s_store_name, s_company_name,
d_moy) tmp1
where case when (avg_monthly_sales <> 0) then (abs(sum_sales -
avg_monthly_sales) / avg_monthly_sales) else null end > 0.1
order by sum_sales - avg_monthly_sales, s_store_name
[_LIMITC];
```

FIG. 10.32: Consulta 89 original do TPC-DS expressa na linguagem SQL.

```
select i_category, i_class, i_brand, s_store_name, s_company_name,
d_moy, sum(ss_sales_price) sum_sales
from item, store_sales, date_dim
where ss_item_sk = i_item_skand
ss_sold_date_sk = d_date_skand
d_year in (2000)and ((i_category in ('Music','Home','Sports')and i_class in
('country','rugs','fishing') ) or (i_category in ('Women','Men','Children')and
i_class in ('swimwear','shirts','pants')))
group by i_category, i_class, i_brand, s_store_name, s_company_name,
d_moy;
```

FIG. 10.33: Consulta 89 alterada do TPC-DS expressa na linguagem SQL.

```
Define HOUR= text({"20",1},{"15",1},{"16",1},{"8",1});
Define DEPCNT=random(0,9,uniform);
define _LIMIT=100;

[_LIMITA] select [_LIMITB] count(*)
from store_sales, household_demographics, time_dim, store
where ss_sold_time_sk = time_dim.t_time_sk
and ss_hdemo_sk = household_demographics.hd_demo_sk
and ss_store_sk = s_store_sk
and time_dim.t_hour = [HOUR]
and time_dim.t_minute >= 30
and household_demographics.hd_dep_count = [DEPCNT]
and store.s_store_name = 'ese'
order by count(*)
[_LIMITC];
```

FIG. 10.34: Consulta 96 original do TPC-DS expressa na linguagem SQL.

```
select count(*)
from store_sales ,customer1,time_dim1
where ss_sold_time_sk = time_dim1.t_time_sk
and ss_customer_sk = customer1.c_customer_sk
and time_dim1.t_hour = 12
and time_dim1.t_minute >= 30
and customer1.hd_dep_count = 3
and store_sales.s_store_name = 'ese';
```

FIG. 10.35: Consulta 96 alterada do TPC-DS expressa na linguagem SQL.

```
Define YEAR=random(1998,2002,uniform);
Define SDATE=date([YEAR]+"-01-01",[YEAR]+"-07-01",sales);
Define CATEGORY=ulist(dist(categories,1,1),3);

select i_item_id, i_item_desc, i_category, i_class, i_current_price,
sum(ss_ext_sales_price) as itemrevenue,
sum(ss_ext_sales_price)*100/sum(sum(ss_ext_sales_price)) over
(partition by i_class) as revenueratio
from store_sales, item, date_dim
where
ss_item_sk = i_item_sk
 and i_category in ('[CATEGORY.1]', '[CATEGORY.2]', '[CATEGORY.3]')
 and ss_sold_date_sk = d_date_sk
and d_date between cast('[SDATE]' as date)
and (cast('[SDATE]' as date) + 30 days)
group by i_item_id, i_item_desc, i_category, i_class, i_current_price
order by i_category, i_class, i_item_id, i_item_desc, revenueratio;

 select i_item_id , i_item_desc, i_category, i_class, i_current_price ,
sum(ss_ext_sales_price) as itemrevenue
from store_sales ,item ,date_dim
where ss_item_sk = i_item_sk
and i_category in ('Music','Home','Sports')
and ss_sold_date_sk = d_date_sk
and d_date between '2002-01-01'and '2002-01-31'
group by i_item_id ,i_item_desc ,i_category ,i_class ,i_current_price;
```

FIG. 10.36: Consulta 98 original do TPC-DS expressa na linguagem SQL.

```
 select i_item_id , i_item_desc, i_category, i_class, i_current_price ,
sum(ss_ext_sales_price) as itemrevenue
from store_sales ,item ,date_dim
where ss_item_sk = i_item_sk
and i_category in ('Music','Home','Sports')
and ss_sold_date_sk = d_date_sk
and d_date between '2002-01-01'and '2002-01-31'
group by i_item_id ,i_item_desc ,i_category ,i_class ,i_current_price;
```

FIG. 10.37: Consulta 98 alterada do TPC-DS expressa na linguagem SQL.