Unioeste - Universidade Estadual do Oeste do Paraná
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
Colegiado de Ciência da Computação
Curso de Bacharelado em Ciência da Computação

Análise de Desempenho entre PostgreSQL e MonetDB para Ambientes OLAP Utilizando TPC-H

Letícia Torres

CASCAVEL
2017

LETÍCIA TORRES

ANÁLISE DE DESEMPENHO ENTRE POSTGRESQL E MONETDB PARA AMBIENTES OLAP UTILIZANDO TPC-H

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação, do Centro de Ciências Exatas e Tecnológicas da Universidade Estadual do Oeste do Paraná - Campus de Cascavel

Orientador: Prof. Dr. Clodis Boscarioli

CASCAVEL
2017

LETÍCIA TORRES

ANÁLISE DE DESEMPENHO ENTRE POSTGRESQL E MONETDB PARA AMBIENTES OLAP UTILIZANDO TPC-H

Monografia apresentada como requisito parcial para obtenção do Título de Bacharel em Ciência da Computação, pela Universidade Estadual do Oeste do Paraná, Campus de Cascavel, aprovada pela Comissão formada pelos professores:

Prof. Dr. Clodis Boscarioli (Orientador)
Colegiado de Ciência da Computação,
UNIOESTE

Gustavo Rezende Krüger (Co-orientador)
Orbit Sistemas

Prof. Dr. Marcio Seiji Oyamada
Colegiado de Ciência da Computação,
UNIOESTE

Bruno Eduardo Soares
Origammi Soluções Digitais

Cascavel, 28 de maio de 2018

DEDICATÓRIA

AGRADECIMENTOS

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

BD	Banco de Dados
TPC	*Transaction Processing Performance Council*
TPC-H	TPC *Benchmark* H
DSS	Decision Support Systems
DW	*Data Warehouse*
OLAP	*On-Line Analytical Processing*
OLTP	*On-Line Transaction Processing*
SGBD	Sistemas Gerenciadores de Banco de Dados
SGBDR	Sistemas Gerenciadores de Banco de Dados Relacional
SF	Factor Scale
PK	*Primary Key*
FK	*Foreign Key*
RF	*Refresh Function*

Lista de Símbolos

Q_i	Consulta, onde $1 \leqslant i \leqslant 15$
S	Número de sessões de consulta do Teste de Vazão
s	Sessão, onde $1 \leqslant s \leqslant S$
RF_j	*Refresh Function* – Função de Atualização
T_s	Tempo em segundos da execução de todo o processo do Teste de Vazão

Sumário

Resumo

Palavras-chave: OLAP, TPC-H, SGBD, Data Warehouse, Benchmark

Capítulo 1

Introdução

Vivencia-se atualmente uma economia caracterizada por uma rápida e constante mudança de mercado e oportunidades de negócio, tal que se tornou essencial às empresas a tomada de decisão correta e de forma rápida baseada em alguma decisão de negócio. Essas decisões são tomadas com base na análise de situações passadas e presentes de uma empresa, ou seja, com base nos dados armazenados por ela. Além disso, uma boa decisão também se utiliza da análise de mercado e predições.

Sob a ótica operacional, de acordo com Wremble e Koncilia [1], os dados de uma empresa são persistidos em sistemas de armazenamento de dados que podem ser heterogêneos, autônomos, e geograficamente distribuídos. Estas características diminuem a eficiência no acesso e processamento dos dados. A gestão de uma empresa requer, no entanto, uma visão abrangente de todos os seus aspectos, exigindo acesso eficiente a todos os dados de interesse. Por este motivo, a capacidade de integrar informações de várias fontes de dados é crucial para uma boa decisão de negócios [1].

Para se obter êxito em uma decisão de negócio é trivial (i) recuperar os dados mantidos por uma empresa; e sobretudo (ii) desenvolver um ambiente de análise cujo objetivo deve ser não apenas informar o significado destes dados, mas sim especular cenários sobre eles com questionamentos como *"e se"* ou *"por quê?"* [2]. Segundo Chaudhuri e Dayal [3] dois elementos são essenciais, dadas as condições anteriores, para uma boa decisão de negócio: *Data Warehouses* (DWs), responsáveis pelo armazenamento homogêneo de dados oriundos de sistemas heterogêneos, e recuperação destes dados; e aplicações OLAP (*On-Line Analytical Processing*).

De acordo com a literatura, existe uma série de princípios que devem ser seguidos ao projetar e implementar um ambiente OLAP dentro de um DW, destacando-se a rapidez com que os dados

devem ser recuperados e processados no DW. Este princípio é considerado fundamental na maior parte da literatura referente à construção de ambientes OLAP, a exemplo de Codd; Codd e Salley [2], Kimball e Ross [4] e Wremble e Koncilia [1]. Sendo assim, vários aspectos técnicos devem ser considerados sob diferentes pontos de vista. Políticas de *cache* específicas para um servidor de estruturas utilizadas pela aplicação OLAP para armazenar e recuperar dados em memória e o SGBD (Sistema Gerenciador de Banco de Dados) utilizado para o gerenciamento do DW são exemplos de aspectos técnicos que também fazem parte de um ambiente OLAP, e que devem ser considerados.

De acordo com Elmasri e Navathe [5] SGBD são importantes para a manutenção e proteção de um banco de dados por um longo período; e também atuam no processo de recuperação de dados de um DW. Neste contexto, a escolha do SGBD no processo de desenvolvimento de um ambiente de análise para organizações com grande quantidade de dados e que utilizam ferramentas OLAP como auxílio na tomada de decisões é importante.

Duas classes de SGBD podem ser utilizadas para realizar o gerenciamento de um DW: SGBD relacionais tradicionais orientados à linha e uma nova abordagem de SGBD NoSQL que são orientados à coluna (SGBD colunares), que fornecem um melhor desempenho à recuperação de dados [6]. Dada a importância na escolha do SGBD e as diferentes soluções apresentadas por cada um, torna-se útil o uso de um benchmark[1] para a realização de uma análise entre SGBD. Existe uma organização sem fins lucrativos fundada com o objetivo de definir padrões para avaliar o desempenho de transações e de bancos de dados com o uso de benchmarks, o TPC (*Transactional Processing Performance Council*) [8]. Esta organização é responsável pela criação de um benchmark voltado para decisões de negócio, o TPC-H [9].

Para refletir a realidade de uma empresa, o TPC-H propõe um ambiente de análise normalizado. Segundo Bax e Souza [10] existe uma discussão sobre a normalização e a denormalização dos dados de um DW. Algumas empresas utilizam um modelo normalizado e têm como justificativa a flexibilidade e a facilidade em situações de manutenção do DW. Por outro lado, existem empresas que utilizam modelos denormalizados e têm como justificativa o ganho de desempenho nas consultas, visto que um modelo denormalizado tende a diminuir o número de tabelas e, por consequência, as operações de junção entre tabelas.

[1] Ferramentas utilizadas para medir e validar o desempenho de alguma tecnologia sob condições de avaliação [7]

O objetivo deste trabalho é a realização de um estudo comparativo entre SGBD, relacionais orientados à linha e orientados à coluna, como gerenciadores de DWs em ambientes OLAP. A comparação será realizada utilizando-se o benchmark TPC-H em sua proposta original, um modelo de banco normalizado; além de uma adaptação para um modelo DW denormalizado. Desta maneira, será possível avaliar os SGBD selecionados de acordo com o modelo do DW, não apenas no contexto original proposto pelo benchmark. Para alcançar tal objetivo, é necessária a execução de uma série de tarefas. A primeira parte engloba o desenvolvimento dos dois ambientes de análise, o modelo padrão proposto pelo TPC-H e o modelo denormalizado, adaptado do TPC-H. Após, são gerados os dados para popular os SGBD e as consultas para as questões de negócio propostas pelo TPC-H; então é feita a população dos dados gerados nos SGBD. A segunda parte consiste na execução das consultas para cada um dos ambientes de análise, sendo que para o ambiente adaptado as consultas devem ser escritas de acordo com as alterações efetuadas. E por fim a aplicação do cálculo de desempenho proposto pelo TPC-H para os dois ambientes.

A organização do documento segue da seguinte forma:

- **Capítulo 2**: apresenta detalhes sobre recuperação de informações, explicando os conceitos de *Data Warehouses*, ambientes OLAP e a conexão entre ambos.

- **Capítulo 3**: apresenta uma breve descrição da motivação no uso de SGBD como gerenciadores de DW, bem como a descrição do modelo relacional, suas limitações e uma introdução a NoSQL.

- **Capítulo 4**: apresenta detalhes de como e porquê surgiram os SGBD NoSQL, assim como a descrição do modelo colunar.

- **Capítulo 5**: apresenta detalhes sobre o benchmark TPC-H, abrangendo informações sobre o ambiente normalizado proposto pelo próprio benchmark; bem como sobre o ambiente denormalizado proposto neste trabalho. É também apresentada a metodologia utilizada para medição e análise do desempenho dos SGBD.

- **Capítulo 6**: é apresentado o estudo comparativo entre os SGBD PostgreSQL e MonetDB, dividido em cenários conforme o fator de escala, bem como a discussão dos resultados obtidos.

- **Capítulo 7**: são discorridas as conclusões obtidas a partir dos resultados obtidos no experimento, assim como trabalhos futuros.

Capítulo 2

Recuperação de Informação

Um ativo importante em qualquer organização é a informação. Segundo Kimball e Ross [4] essa informação é mantida sob duas formas: sistemas de banco de dados operacionais, onde a informação é armazenada; e DWs, onde ela é recuperada. Em sistemas operacionais geralmente os usuários lidam com o mesmo registro e realizam a mesma tarefa exaustivamente sob uma única informação, permanecendo no domínio das transações; enquanto que em um DW pode-se ver o progresso da organização utilizando dados armazenados continuamente, de forma otimizada para a recuperação de dados. Também, são formuladas perguntas com a finalidade de responder a alguma questão de negócio, como *"quantos pedidos foram recebidos pelo fornecedor X no período de tempo Y?"*, ou *"qual foi o impacto no número de vendas ao mudar o formato de envio de A para B?"*. Para responder questões dessa natureza não é viável lidar com dados de forma individual, mas sim recuperar um conjunto de dados a fim de formular uma resposta.

2.1 *Data Warehouses* e Aplicações OLAP

De acordo com Inmon [11], DWs são base de todos os Sistemas de Decisão de Suporte (*Decision Support Systems* – DSS). DSS são tecnologias utilizadas para decisões de negócio e solução de problemas, e incluem componentes que realizam gerenciamento de banco de dados e que permitem uma interação com o usuário de forma a simplificar consultas e geração de relatórios [12]. DWs foram as primeiras ferramentas a surgirem como solução para o suporte à decisão de negócio, integrando dados de diferentes bancos de dados operacionais [11, 4].

De forma geral, um DW é um repositório de dados capaz de fornecer rapidamente informa-

ções consistentes e cruciais para a tomada de decisão de uma organização, de tal forma que essa informação possa ser acessada de maneira intuitiva e legível pelo usuário, a fim de combinar diferentes informações entre os dados armazenados [4]. Deve também se adaptar a possíveis mudanças, sejam elas mudanças comerciais, mudanças nos dados ou na tecnologia. Inmon [11] define um DW como: uma coleção de dados não-volátil, ou seja, que não muda após inserida no *warehouse*; orientado ao assunto principal da organização; integrado e variante no tempo para que seja mantido um histórico a fim de analisar situações passadas. Do ponto de vista estrutural, Wremble e Koncilia [1] definem um DW como uma base de dados homogênea, local e centralizada.

Para analisar os dados de um DW além de implementá-lo é necessário que alguma aplicação leia seu conteúdo e apresente-o de forma gráfica e intuitiva ao analisador. Aplicações OLTP (*On-Line Transactional Processing*) são utilizadas por bancos de dados operacionais e operam transações atômicas e isoladas de forma repetitiva, que correspondem ao dia-a-dia de uma organização [3]. DWs trabalham com suporte à decisão e são intensivos à consultas *ad hoc* complexas, que acessam milhões de registros. Sendo assim, os dados históricos, a taxa de vazão de uma consulta e o tempo de resposta são mais importantes que pequenas transações. À aplicação aceita por um DW dá-se o nome de OLAP, cujo objetivo, de acordo com Codd; Codd e Salley [2], é identificar tendências, padrões de comportamento e anomalias, bem como relações em dados aparentemente não relacionados. Os resultados dessas análises servem de base para tomada de decisões de negócio. Portanto, DWs e aplicações OLAP são componentes chave para a construção de um ambiente de análise.

Dentro do domínio de um DW existem diversos componentes que realizam funções específicas a fim de construir um ambiente de *warehouse* desde a obtenção dos dados de fontes externas e sistemas operacionais de bancos de dados, até o acesso a esses dados por meio do DW por alguma consulta analítica definida sob uma aplicação OLAP. Para entender esta arquitetura fim-a-fim, ilustrada na Figura 2.1 adaptada de Kimball e Ross [4], é necessário compreender alguns componentes e conceitos que formam um DW [4]:

- **Sistemas de Fonte Operacional**: possuem detalhes sobre as transações do negócio, correspondendo a ambientes OLTP. Engloba os dados que irão estruturar as informações do DW, portanto se encontram externos ao *warehouse*. Podem ser tanto sistemas de banco

de dados ou alguma outra fonte de dados, como um documento no formato XLS, CSV, TXT; e sistemas CRM.

- ***Staging Area***: compreende tanto uma área de armazenamento temporária quanto um conjunto de processos denominado ETL (*extract-transformation-load*). De forma geral é uma área a qual os usuários não têm acesso, onde os dados são traduzidos para algo que possa ser enviado de maneira compatível ao *warehouse* e não se trabalha diretamente sobre os dados transacionais. Quanto aos processos ETL, a fase de Extração (*Extraction*) consiste na leitura da fonte de dados, transferindo o conteúdo necessário para a *staging area*; após essa extração pode ser necessário realizar uma "limpeza"nos dados; unir dados de diferentes fontes; tratar duplicatas e atribuir chaves do *warehouse*. A isto dá-se o nome de Transformação (*Transformation*). A última fase, fase de Carregamento (*Load*), é responsável por carregar, ou popular, os dados na área de estruturação de dados do DW.

- **Estruturação de Dados**: trata de como os dados serão organizados, armazenados e disponibilizados para consultas de usuários, relatórios e outras aplicações. No que tange à comunidade empresarial, a fase de apresentação de dados *é* o DW em si, pois corresponde ao que pode ser acessado via ferramentas de acesso a dados. A etapa de estruturação é comumente definida como sendo um conjunto de *data marts*. *Data marts* são subconjuntos do total de informações de um DW, cada qual representando os dados de um determinado assunto, departamento, ou processo de negócio. Nesta fase é definida a modelagem conceitual do ambiente de análise do DW.

- **Ferramentas de Acesso aos Dados**: são formas de aplicar uma consulta, dentro de aplicações OLAP, aos dados organizados na fase de estruturação. Pode ser uma consulta *ad hoc* ou algo mais complexo, como consultas aplicadas à mineração de dados.

Existe uma discussão acerca da modelagem conceitual da área de apresentação de dados de um ambiente de análise nos DWs [13]. Segundo Sen e Sinha [13] as duas técnicas de modelagem mais utilizadas são a ER (Entidade-Relacional) e a Dimensional. A primeira segue o padrão de modelagem para ambientes OLTP, que traduz a modelagem ER para um esquema relacional em seguida normalizando-o geralmente até a Terceira Forma Normal (3NF) [4]. O

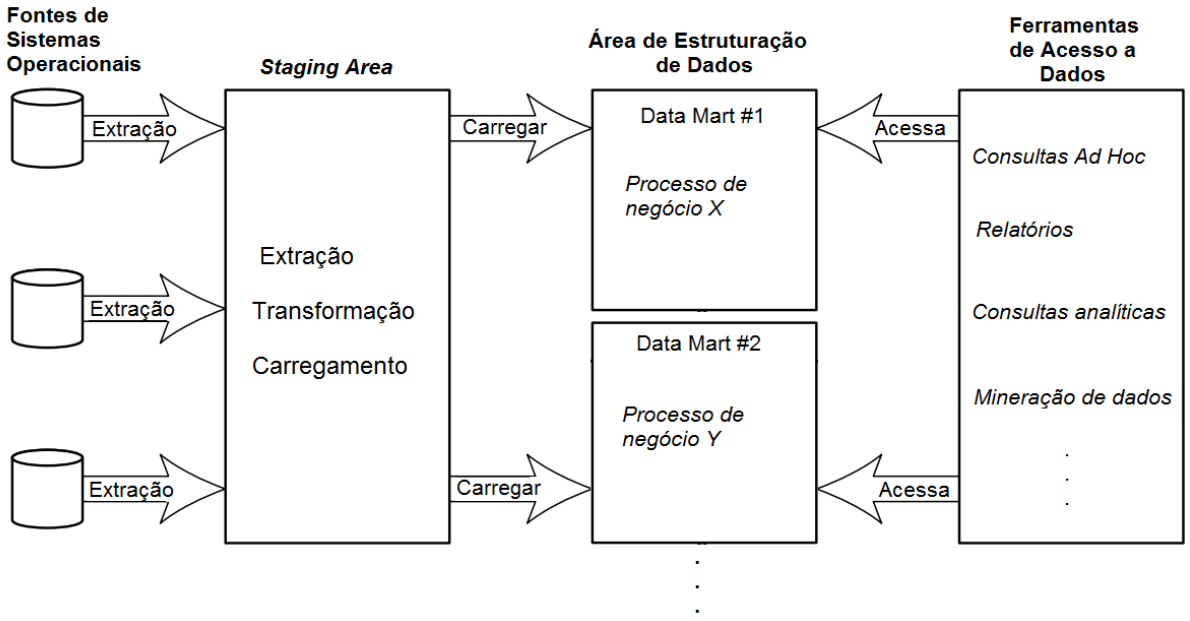

Figura 2.1: Arquitetura de um *Data Warehouse*.

modelo Dimensional, ou multidimensional, por sua vez, evita atingir o mesmo nível de normalização da modelagem ER.

O modelo dimensional é composto por tabelas denominadas Tabelas Fato e Tabelas Dimensão [4], conhecido comumente como modelo *Star Join*, ou apenas *Star* [13]. Uma Tabela Fato é a principal tabela do modelo Dimensional, contemplando atributos responsáveis por determinar as métricas de negócio. Em sua maioria são atributos numéricos, relacionados à *quantidade*; e aditivos, visto que uma consulta em um DW pode retornar até milhares de tuplas, tornando interessante o conhecimento de informações como o total de um atributo dada alguma questão de negócio. As Tabelas Fato são auxiliadas pelas Tabelas Dimensão no que concerne à descrição textual das questões de negócio. É comum estas tabelas terem de 50 a 100 atributos, e que estes atributos sejam responsáveis pelas restrições de uma consulta, sendo também comumente utilizados em agrupamentos. Todas as Tabelas Fato tem duas ou mais chaves estrangeiras (*Foreign Keys* – FKs) relacionadas às chaves primárias (*Primary Keys* – PKs) das Tabelas Dimensão, como mostra o exemplo da Figura 2.2, onde a tabela Vendas corresponde à uma Tabela Fato e as demais à Tabelas Dimensão. Note que esta figura também faz referência a um modelo *Star*.

Mesmo que a modelagem Dimensional não atinja a normalização 3NF, modelos *Star* podem ser trabalhados de forma a oferecer suporte à hierarquia de atributos das Tabelas Dimensão,

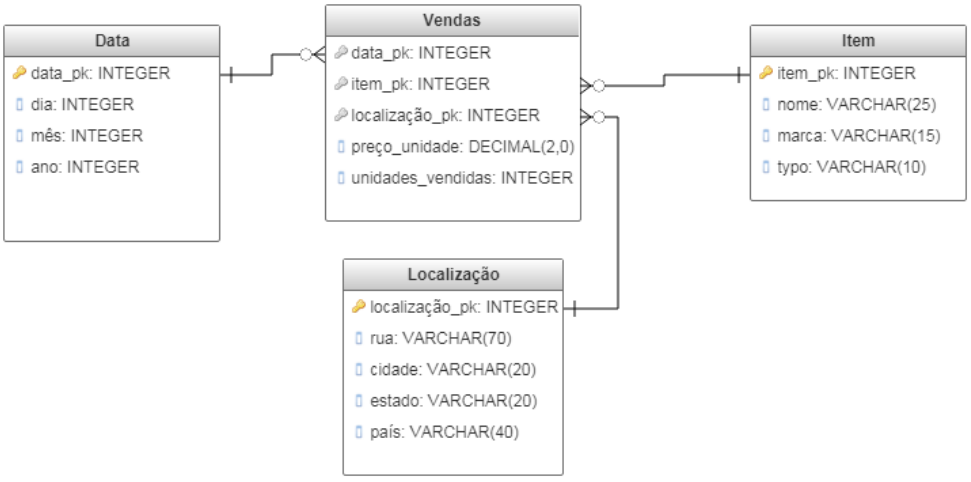

Figura 2.2: Exemplo de esquema *Star* com tabelas Fato e Dimensão

permitindo que estas tenham "Tabelas Subdimensão". A esse refinamento se dá o nome de *Snow Flake* [5]. Embora tenham uma estrutura mais simplificada, segundo Levene e Loizou [14] a escolha do uso de esquemas *Snow Flake* se dá por serem um esquema intuitivo, de fácil entendimento, passíveis à otimização de consultas, e de fácil extensão – uma vez que pode-se adicionar atributos às tabelas sem interferir em programas já existentes. Uma possível adaptação de um modelo *Star* para *Snow Flake* é como mostrado na Figura 2.3, no qual foi adaptado o modelo da Figura 2.2, adicionando a Tabela Subdimensão Cidade.

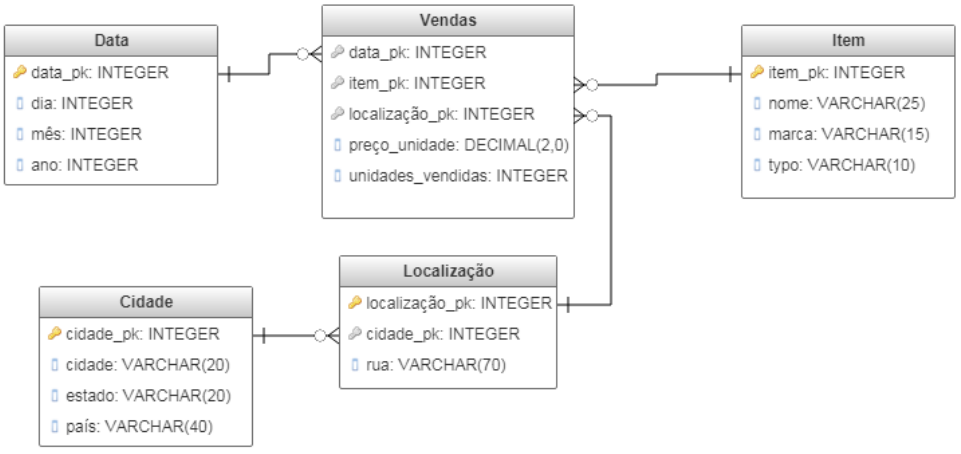

Figura 2.3: Exemplo de esquema *Snow Flake* adaptado da Figura 2.2

Para que possa ser construído um DW e aplicada a modelagem acima descrita, é necessário alguma ferramenta que possa fazer o gerenciamento deste DW. De acordo com Elmasri e Na-

vathe [5] um banco de dados pode ser gerenciado por sistemas que facilitem este processo de gerenciamento no banco de dados. Estes sistemas são conhecidos como Sistemas Gerenciadores de Banco de Dados, ou SGBD.

Capítulo 3

Sistemas Gerenciadores de Banco de Dados

Antes de se ter um sistema voltado para gerenciamento de um banco de dados, eram utilizados para persistência de dados o sistema de arquivos. Apesar de simples, esta abordagem apresentava alguns problemas por não apresentar suporte à redundância de informações; não garantir integridade de dados; falta de segurança; e o acesso e gerenciamento dos dados depende de programas e aplicativos, fazendo com que seja necessário criar um novo aplicativo a cada requisição de dados diferente. Outro problema crítico é não se ter informação de relacionamento entre arquivos diferentes.

Como forma de manter o gerenciamento de dados independente de aplicações e programas bem como solucionar os demais falhas do sistema de arquivos foram criados os Sistemas Gerenciadores de Bancos de Dados, os SGBD. De acordo com Elmasri e Navathe [5], os SGBD são uma coleção de programas para criação e manutenção de um banco de dados, que facilita a definição, construção, manipulação e compartilhamento de dados entre usuários e aplicações.

Dentre as vantagens que os SGBD trouxeram em detrimento ao sistema de arquivos estão:

- **Controle de redundância**, para que não seja permitido duplicação de dados, pois isto causaria desperdício na capacidade de armazenamento.
- **Restrição de acesso aos usuários**, pois não será permitida manipulação do banco a todos os usuários, ou funcionários de uma empresa por exemplo.
- **Execução de consultas eficiente** através do uso de índices, normalmente implementadas utilizando hash ou árvores, para que o acesso ao disco seja mais rápido.

- **Restrições de integridade**, a fim de garantir que (i) dados não sejam inseridos de forma inconsistente de acordo com o tipo de atributo definido; (ii) as relações entre entidades sejam efetuadas e (iii) restrições de chave sejam mantidas.
- **Persistência de dados**, para garantir que os dados serão inseridos e de fato armazenados no modelo.
- ***Backup*** de dados periodicamente para evitar problemas caso aconteça alguma perda no banco e posterior **restauração**, para recuperar uma imagem do último backup feito no banco.

Existem várias classes de SGBD, conforme sua estruturação e a forma como manipulam os dados, entre as mais conhecidas estão o modelo relacional, objeto-relacional, orientado a objetos e a abordagem mais recente NoSQL.

3.1 SGBD Relacional

O modelo mais utilizado de SGBD é o relacional, ou SGBDR (SGBD Relacional). Ele foi conceituado por Codd [15] em 1970 em um artigo no qual são expostas as vantagens de um modelo relacional em detrimento a um modelo de redes.

Um modelo relacional define através de um conjunto de tabelas entidades que representam objetos do mundo real, cada qual com seus atributos. Esse conjunto de atributos é denominado registro do banco de dados, representando uma tupla na tabela. Assim como no mundo real, objetos devem estar também relacionados, e para tal SGBDR utilizam-se de chaves.

Em um SGBDR além de definir relacionamentos, as chaves também garantem integridade entre os dados. Existem dois tipos de chaves, uma que define a unicidade de um atributo, ou seja, assegura que não haverão registros repetidos no banco, chamada de chave primária (*primary key*, ou PK); e outra que garante integridade no relacionamento entre duas entidades referenciando uma tabela na outra, que é a chave estrangeira (*foreign key*, ou FK).

Tomando como exemplo uma empresa fictícia, são criados dois objetos reais que precisam ser representados sob a forma relacional, *funcionário* e *projeto*. Nesta emprese funcionários possuem nome, CPF, sexo, salário, um supervisor e trabalham em um ou mais projetos. Esses projetos também possuem atributos: nome, código e o departamento no qual foram criados.

Apenas com estas informações duas tabelas já são definidas no banco, funcionário e projeto, bem como seus atributos, como ilustra a Figura X.

Como descrito, um funcionário trabalha em um ou mais projetos e ele precisa ainda registrar o número de horas trabalhadas nestes projetos. Desta forma é preciso de alguma forma relacionar funcionário com os projetos. Neste exemplo cabe a criação de uma nova entidade responsável unicamente por relacionar estes dois objetos e ainda armazenar o número de horas – veja que o número de horas não cabe na tabela de funcionários nem na de projetos.

É nesta relação que são explorados os conceitos de chaves em um banco de dados. Para que a entidade que relaciona funcionário e projeto tenha informações de que funcionário trabalha em qual projeto é necessário acessar um número, ou código, definido para diferenciar todos os funcionários da empresa bem como o código do projeto e relacioná-los. Para o funcionário uma PK válida é o número do CPF e para o projeto o próprio atributo código. Essa relação pode ser nomeada unindo as entidades que relaciona, nesse caso *funcionário_projeto*, ou quando faz sentido pode ser nomeada de acordo com a função que realiza, neste caso como um funcionário trabalha em um projeto a relação pode ser *trabalha_em*.

O esquema que representa estas relações é como mostrado na Figura Z. Nela podemos ver as chaves primárias, em destaque, nas relações *funcionário* e *projeto*, e seus valores como chave estrangeira na relação *trabalha_em*.

Modelos de bancos de dados prezam, sempre que possível, pela consistência de dados, principalmente se tomar operações transacionais como transações bancárias, que necessitam de um alto grau de consistência após operações de inserção e atualização, para que não haja perda de valores ou que esses ainda façam sentido no sistema. Por essa razão SGBDR seguem o conceito ACID (Atomicidade, Consistência, Isolamento e Durabilidade).

Modelos relacionais também seguem um alto grau de normalização nos dados a fim de evitar redundância e confusão entre atributos que podem ser desmembrados em outras entidades. Essa normalização é realizada normalmente até a Terceira Forma Normal (3NF), e o esquema final conterá um número maior de tabelas que o original devido ao desmembramento de atributos. Essa quantidade maior de tabelas acarreta no aumento de junções necessárias para recuperar atributos entre entidades relacionadas, ou seja, no aumento do acesso a entidades e no processamento de tuplas.

Além de diminuir o desempenho na recuperação de dados, um esquema normalizado causa um forte acoplamento entre tabelas, o que tornaria inviável o particionamento dos dados do banco em *n* servidores como forma de escalar estes dados conforme seu volume aumenta (mais detalhes no Capítulo 4), como é o caso de um ambiente analítico. Uma solução trivial para este problema é a denormalização de dados, porém para um SGBDR essa solução pode não ser viável, visto que sua implementação se dá seguindo a modelagem relacional.

Exemplos de SGBDR são PostgreSQL [16], MySQL [17], Microsoft SQL Server [18], SQ-Lite [19], MariaDB [20] e Oracle [21].

3.2 SGBD NoSQL

Bancos com o objetivo de suprir problemas apresentados por SGBDR começaram a emergir entre 2004 e 2007 visando alta escalabilidade, recuperação e armazenamento de grandes volumes de dados de forma eficiente, e ao mesmo tempo diminuir custos com hardware [22]. Esses bancos levaram o nome de NoSQL, porém isso não significa que não utilizam SQL para manipulação de dados.

Detalhes sobre bancos de dados NoSQL são encontrados no Capítulo 4, no qual também é descrito o modelo colunar, utilizado por satisfazer os requerimentos de um ambiente de DW.

Capítulo 4

NoSQL

O desenvolvimento de novas aplicações e surgimento de novas soluções para sistemas gerou crescimento no volume de dados de maneira acelerada. Com isso cresce também o número de usuários, necessitando portanto escalar o BD. Existem duas soluções para escalar um sistema: o escalonamento vertical, que consiste em um *upgrade* do servidor no qual o banco está hospedado; e o horizontal, aumentando o número de servidores e distribuindo o banco [23, 24].

Existem algumas desvantagens no *upgrade* de um sistema. O BD pode superar a capacidade da melhor configuração disponível no mercado, e ainda é caro por requerer a aquisição de uma configuração melhor. Mesmo considerado mais complexo, o escalonamento horizontal é mais viável. Entre as soluções apresentadas no particionamento horizontal estão o particionamento funcional e o *sharding* [23].

O particionamento funcional consiste em fragmentar os dados de acordo com a forma como são utilizados dado um contexto. Um esquema com quatro entidades, *usuários*, *produtos*, *clientes* e *endereço* pode ser distribuído em quatro servidores, um para cada entidade. Entidades que são utilizadas somente para leitura podem ser separadas de entidades onde dados são escritos, caracterizando outro exemplo de particionamento funcional. O problema com esta estratégia está no acoplamento entre entidades: se duas ou mais entidades estiverem relacionadas elas deverão estar no mesmo servidor, caso contrário não será possível atribuir as restrições de chave àquele relacionamento [23]. Tomando o exemplo acima com as quatro entidades, supondo que *cliente* e *endereço* estejam relacionadas estas devem ser armazenados no mesmo servidor.

A segunda estratégia, o *sharding*, consiste em dividir os dados do banco utilizando algum critério de separação de dados que não seja limitado à funcionalidade das entidades. Soluções como particionamento com base em *hash* ou listas podem ser aplicadas. Considerando dez

servidores e uma chave primária auto-incremental, a função de *hash* pode tomar o módulo da chave primária pelo total de servidores como critério de seleção da partição na qual o dado será inserido. Utilizando uma lista é possível definir valores para os servidores e distribuir os dados de acordo com esses valores. Por exemplo, as linguagens C++, Java e C# poderiam ser inseridas em uma partição destinada à linguagens orientadas a objeto.

O *sharding* enfrenta problemas com junções entre dados, visto que os dados devem ser recuperados de diferentes partições e a maioria dos SGBDR não oferece suporte a chaves estrangeiras sob diferentes servidores, restando ao desenvolvedor tratar isso no código da aplicação [23]. Uma solução para tais problemas é a denormalização de dados para que o número de junções diminua ou, no melhor dos casos e quando possível, seja zero. Contudo, isso é um problema para modelos relacionais, visto que trabalham sobre uma modelagem normalizada de dados. Além disso e de não se adaptarem à escalabilidade horizontal, são complexos e o fato de trazerem todas as informações de uma entidade sob a forma de tuplas causa lentidão em um ambiente analítico na recuperação de dados, cujas consultas percorrem o banco visando atributos específicos, processando somente o necessário.

Outro problema crítico ao não utilizar a escalabilidade horizontal está na disponibilidade dos dados. Enquanto um banco se apoiar na escalabilidade vertical, qualquer queda no servidor acarretará na queda total no sistema de armazenamento, ao passo que quando se trabalha com servidores em paralelo a queda em uma máquina não trará prejuízos no conjunto todo. Assim, soluções como melhorar o hardware do servidor continuam sendo desvantajosas. Além da escalabilidade, disponibilidade de dados foi um dos argumentos utilizados por um dos engenheiros da rede social Twitter ao migrar do MySQL para o Cassandra, um SGBD NoSQL. Em 2008 a rede ficou fora do ar por 84h [25].

Com o intuito de suprir tais problemas de escalabilidade, disponibilidade e recuperação rápida de dados, entre 2004 e 2007 os SGBD NoSQL começaram a ganhar destaque com o surgimento das bases de dados BigTable da Google [26], e Dynamo da Amazon [27]. O termo NoSQL, embora a princípio pareça indicar total independência de SQL, significa *Not Only SQL*, "Não Apenas SQL". Também, ele não descreve um único tipo de SGBD, mas sim uma classe de modelos, cada qual com suas propriedades. As mais conhecidas são quatro:

- **Orientado a Grafos**, que se utiliza da Teoria dos Grafos para estruturar seus dados. Um

exemplo desta classe é o Neo4j [28].

- **Orientado a Chave-Valor**, que armazena os dados de forma similar a uma tabela *hash*, com uma chave referenciando um valor, ou tipo de dado. Exemplos são o Project Voldemort [29], DyanmoDB [30], o Riak [31] e o Redis [32].

- **Orientado a Documento**, uma versão melhorada do Chave-Valor, no qual os valores são armazenados como documentos através de estruturas complexas como JSON e XML. Exemplos são o MongoDB [33] e o CouchDB [34].

- **Orientado a Colunas**, ou modelo colunar, que utiliza tabelas como armazenamento de entidades, porém não agrupa os dados sob forma de tuplas, e sim colunas. Exemplos são o MonetDB [35], C-Store [36], BigTable [37] e Cassandra [38].

De maneira geral, as vantagens no uso de um SGBD NoSQL estão no rápido processamento de um grande volume de dados, flexibilidade para expansão e baixo custo de escalabilidade. Devido ao vasto número de SGBD dentro de cada classe de NoSQL, esses bancos também podem ser classificados de acordo com o Teorema CAP (do inglês *Consistency, Availability, Partition tolerance*). Segundo Eric Brewer [39, 40], um sistema distribuído não é capaz de conciliar consistência, disponibilidade e tolerância a partição de dados simultaneamente, tendo que optar por apenas dois destes. SGBDR prezam por disponibilidade e consistência de dados seguindo o ACID, porém a preocupação com consistência pode tornar a manipulação de dados lenta. Visto que o movimento NoSQL surgiu com a intenção de melhorar a escalabilidade e disponibilidade de dados, e tornar a manipulação destes mais rápida, a maioria deles trabalha com os atributos de disponibilidade e tolerância à partição. Essa configuração assume um conceito diferente do ACID para bancos NoSQL, e Pritchett [23] propôs um teorema diferente do ACID, mais otimista, onde a consistência é relaxada, o Teorema BASE (*Basically Available, Soft State, Eventual Consistency*).

O Teorema BASE assume que a consistência em um banco de dados está em estado de fluxo, ao contrário do ACID que força a consistência a cada operação, sendo este considerado pelo autor um método pessimista. Neste cenário a disponibilidade é garantida devido à tolerância a partição, fazendo com que a falha de um servidor não cesse o funcionamento de todo o sistema

– por exemplo, caso uma partição falhe em um sistema rodando sobre dez servidores, apenas 10% dos dados estarão indisponíveis e somente os usuários daquela partição serão afetados.

4.1 SGBD Colunar

Dentre as categorias de bancos de dados NoSQL a que melhor se adequa aos propósitos analíticos é a colunar. Sistemas colunares armazenam seus dados em colunas que representam atributos das entidades, fazendo com que apenas os atributos necessários sejam lidos [41], o que diminui o tempo de acesso ao disco [42, 43]. Cada uma destas colunas pode armazenar seus valores utilizando o par *chave, valor* [44, 41], sendo esta uma das formas de implementação de um sistema colunar (descrita na Seção 4.2 deste capítulo). A Figura X ilustra de forma simples, embora apenas visual, a diferença principal entre um armazenamento utilizando linhas e outro utilizando colunas.

Matei [42] cita algumas vantagens de sistemas colunares:

- **Melhor desempenho**, pela forma com que os dados são armazenados e o uso de índices visando o armazenamento ao invés de localização de registros, resultando em menos operações de entrada e saída.
- **Rápidas operações de agrupamento**, bastante utilizadas em ambientes OLAP, visto que valores de um mesmo atributo são armazenados consecutivamente. Bem como operações matemáticas, como recuperar o maior ou menor valor, soma e média.
- **Alta compressão de dados** pode ser alcançada, devido às chances de se ter valores repetidos para um mesmo atributo serem maiores.

Existem diferentes abordagens para se implementar um SGBD colunar. Essas abordagens levam em conta mudanças na codificação de um banco, apenas na modelagem dele, bem como se será possível atingir níveis altos de compressão. A primeira técnica consiste no **particionamento vertical** dos dados, a segunda em **modificar a camada de armazenamento** e a terceira corresponde a junção de ambas.

4.2 Implementação do Sistema Colunar

Antes de detalhar os métodos de implementação de um BD colunar, é fundamental discutir sobre a reconstrução de tuplas nesse tipo de SGBD. Segundo Abadi et al. [45], a parte lógica e visual de um BD colunar se apresenta da mesma forma que um BD relacional. Isso faz com que a maioria dos SGBD colunares ofereçam uma interface relacional de comunicação, compatível com os padrões.

Como os atributos de um modelo colunar acabam ficando separados em disco um do outro, eles precisam ser unidos novamente em tuplas para que o resultado de uma consulta seja exibido. Existem duas técnicas para realizar essa reconstrução, ou materialização: *early materialization* (EM) e *late materialization* (LM) [45, 46].

A *early materialization*, melhor traduzida como materialização prévia nesse contexto, é a política adotada por BD relacionais, e consiste em adicionar as colunas à tupla conforme a coluna é requisitada na consulta. Essa técnica não leva em conta o predicado da consulta, isto é, considere a consulta a seguir:

```
SELECT nome FROM funcionario
WHERE salario >= 1000 AND sexo='F'
```

A EM irá processar essa consulta e construir uma tupla com os atributos *nome, salário* e *sexo*, com todos os registros armazenados em banco. Considere que os registros são como mostrados na Figura X. Esse método só analisa o predicado após a construção das tuplas. Isso não acontece na LM.

Na LM, ou materialização tardia, é analisado primeiro o predicado e verificado quais valores atendem a esse predicado em cada coluna. No caso da SQL acima primeiro seria analisado o predicado de seleção, nesse caso `WHERE salario >= 1000 AND sexo='F'`, e então através de um operador lógico AND seria possível retornar a posição dos atributos que satisfazem essa seleção, para então construir a tupla a partir do predicado de projeção somente com o atributo *nome*. Em suma, apenas após ter a posição de cada atributo é que a tupla é construída, descartando a reconstrução de tuplas que seriam posteriormente descartadas. A Figura A e B representam o resultado final de cada materialização.

4.2.1 Particionamento Vertical

Considerada a técnica mais simples para implementar um banco colunar, esta técnica realiza a partição dos dados de forma que cada atributo de uma entidade seja armazenado em uma tabela de duas colunas contendo o par <chave/índice, valor do atributo> [41].

A Figura Y mostra que é possível desmembrar o esquema relacional em colunas de atributos. No momento em que uma consulta analítica é realizada, não serão processados dados além dos de fato requeridos e nenhum dado é perdido pois ainda haverá referência entre os atributos através das chaves.

Essa abordagem é a mais simples de se implementar, pois nenhuma mudança é feita na codificação do BD, sendo que a única alteração necessária está a nível da aplicação responsável por executar as consultas, mudando a lógica de acesso aos dados. Isso torna prático para empresas a mudança entre a estratégia relacional e a colunar, pois torna possível o uso de um mesmo SGBD como DW.

Ao mesmo tempo que essa implementação traz praticidade na migração de dados, ela não difere muito da abordagem relacional no que concerne ao desempenho. Em Abadi [46] é mostrado que adaptar dados de um esquema relacional utilizando particionamento vertical não trouxe melhoras no desempenho das consultas. Foi verificado que o custo de junções para unir as colunas é alto conforme a quantidade de atributos selecionados aumenta e não foi possível usufruir eficientemente da compressão de dados.

4.2.2 Modificação na Camada de Armazenamento

A segunda abordagem para implementar um sistema colunar consiste em já modificar o armazenamento do banco. Essa técnica não altera a parte lógica da modelagem do banco, mas a camada física alterando o armazenamento de linha por linha para coluna por coluna. Nessa implementação as chaves não precisam ser repetidas como na abordagem anterior, na qual a primeira coluna de cada tabela consistia no valor de chave da cada atributo. O *i-ésimo* valor de atributo de uma coluna estará associado ao *i-ésimo* atributo das demais colunas e não é desperdiçado espaço em disco com as chaves.

Neste método a reconstrução de tuplas é realizada antes da execução da consulta de fato. Como apenas um determinado número de atributos precisa ser acessado, estes são combinados

seguindo a lógica que o *i-ésimo* atributo de uma coluna está relacionado com o *i-ésimo* atributo das demais. Aqui tem-se uma vantagem no modelo orientado a linhas, por este já armazenar as sob o formato de linhas e não precisar armazenar os dados em ordem.

4.2.3 Modificação na Camada de Armazenamento e Particionamento Vertical

A última, e melhor, abordagem consiste em modificar a camada de armazenamento do BD e a aplicação que processa as consultas [47, 48], assim o processador pode manter os dados sob a forma de colunas sem precisar unir as colunas e construir tuplas. Uma vantagem óbvia desta técnica é a eliminação da reconstrução de tuplas e com isso menos dados precisam ser previamente movimentados, diminuindo o tempo de processamento.

4.3 Compressão de Dados

Uma característica marcante de um SGBD colunar é a alta possibilidade de compressão de dados. Existem diversas formas de comprimir os dados, porém deve-se ater ao fato de que o tempo de compressão e descompressão não deve afetar significativamente o processamento dos dados no SGBD. Para tal, uma solução é utilizar métodos que consigam operar sob os dados ainda comprimidos – ainda nesta seção são apresentados alguns dos algoritmos utilizados para compressão em BD.

Como exemplo, apenas ilustrativo, de como os dados em um sistema colunar podem ser comprimidos de forma mais eficiente que em um relacional, considere a entidade *usuário* a seguir, com os atributos *id, nome, idade* e *sexo*. A primeira coluna corresponde aos índices utilizados no banco.

```
1: 0, Bruno, 50, M
2: 10, João, 23, M
3: 20, Maria, 30, F
4: 90, Ana, 23, F
5: 78, Gabriel, 23, M
```

Um banco relacional armazena estes atributos da forma como foram apresentados acima. Considerando o armazenamento colunar, os mesmos dados seriam armazenados da seguinte forma:

```
1: 0, 2: 10, 3: 20, 4: 90, 5: 78
1: Bruno, 2: João, 3: Maria, 4: Ana, 5: Gabriel
1: 50, 2: 23, 3: 30, 4: 23, 5: 23
1: M, 2: M, 3: F, 4: F, 5: M
```

Note que fica mais claro que como os atributos são armazenados em uma coluna com chave e valor existirão mais chances de repetição de valores em uma mesma coluna. Os atributos *idade* e *sexo* se repetem para alguns registros, possibilitando assim compressão destes dados. Comprimindo estas repetições o nosso modelo é armazenado da forma:

```
1: 0, 2: 10, 3: 20, 4: 90, 5: 78
1: Bruno, 2: João, 3: Maria, 4: Ana, 5: Gabriel
1: 50, 2, 4, 5: 23, 3: 30
1, 2, 5: M, 3, 4: F
```

Westmann et al. [49] consideram que os métodos de compressão em um BD devem ser capazes de se aplicar no banco como um todo, bem como em uma tupla ou a cada atributo de tupla, e serem rápidos em processamento quando há casos em que seja necessário realizar descompressão.

O primeiro método consiste em comprimir a representação de valores inteiros, baseado no algoritmo *null supression* [49, 50]. A ideia é descartar zeros na representação de um inteiro de quatro bytes, por exemplo, ao invés de se representar o número 10 da forma '00000000000000000000000000001010', seriam utilizados apenas quatro bits, '1010'. É necessário entretanto guardar a informação de quantos bits são utilizados para armazenar um número e em alguns casos também é preciso armazenar um bit a mais para o sinal do inteiro. Esse algoritmo pode ser utilizado para comprimir o tamanho de strings no banco quando estas são representadas pelo tipo VARCHAR, visto que para esse tipo de dado o tamanho também é armazenado devido a possibilidade deste variar.

Um método bastante empregado é a Codificação por Dicionário. Nele atributos que possuem um padrão fixo, por exemplo, os números primos de 3 a 19 (3, 5, 7, 11, 13, 17, 19), podem ser representados por um dicionário de 3 bits. Este algoritmo pode ser utilizado para compressão de valores NULL, considerando-o como um possível valor no padrão.

Abadi, Madden e Ferreira [51] utilizam a compressão por dicionário e uma variação do algoritmo *null supression* para implementar o BD C-Store e ainda apresentam dois outros métodos. O primeiro é o *run-length*, que pode ser bem aproveitado em bancos colunares quando atributos

são repetidos sequencialmente e possuem pouca variação. Ele consiste em contar o número de vezes no qual um valor é repetido utilizando a tripla (valor, posição inicial, tamanho). Um exemplo é ilustrado na Figura A com o atributo *especialização*.

O último método apresentado pelos autores é a Codificação por Vetor de Bit, bastante útil quando uma coluna possui número limitado de possibilidades, como linguagens de programação dentro de um dado paradigma ou estados do Brasil. Uma string de bits representa a ocorrência do valor de atributo, onde '1' representa a ocorrência naquela posição e '0' a não ocorrência. Como exemplo, uma coluna com os valores M M F M F F M podem ser representados pela cadeia de bits 1 1 0 1 0 0 1 para o valor M, e 0 0 1 0 1 1 0 para F.

Em suma, ao implementar um banco colunar tanto a aplicação deve ser capaz de entender e conseguir recuperar informações mesmo comprimidas, como a modificação da camada de armazenamento deve ser implementada tal que se possa usufruir de algum método de compressão. Caso contrário um dos principais diferenciais de modelos colunares não será explorado e poderá não haver impacto no desempenho quando trabalhado com uma grande massa de dados, como é a realidade de empresas.

Capítulo 5

TPC *Benchmark* H

A essência de um *benchmark* é identificar e definir os melhores padrões de excelência para produtos, serviços e/ou processos, e então realizar aperfeiçoamentos de forma que esses padrões sejam alcançados [52]. De maneira geral, *benchmarks* se consolidaram como uma ferramenta para melhorar o desempenho de organizações e a competitividade nos negócios [53].

O TPC *benchmark* H é um padrão de decisão de negócio internacionalmente aceito para comparações de desempenho entre SGBDs utilizados em ambientes OLAP criado pela organização TPC. De acordo com a página oficial do TPC[1], esta organização sem fins lucrativos foi fundada com o objetivo de definir padrões para *benchmarks* no processamento de transações e bancos de dados. Ela é responsável pelo desenvolvimento e atualização destes *benchmarks*, bem como pela divulgação dos resultados apresentados por eles. Entre os sócios[2] responsáveis por isto estão as empresas Dell, Hewlet Packard, IBM, Microsoft, Oracle, Intel e Cisco. A lista de todos os sócios pode ser acessada na página oficial da organização.

Alguns trabalhos já foram realizados utilizando o TPC-H [54, 55, 56, 57, 58, 59]. Ngamsuriyaroj e Pornpattana [54] aplicam as consultas do TPC-H no MySQL Cluster a fim de avaliar seu desempenho. O trabalho de Nadee e Ngamsuriyaroj [55] tem uma proposta parecida com o primeiro [54], porém o intuito é avaliar o desempenho de um *cluster* Java EE baseado nas características das consultas do TPC-H. Thanopoulou et al. [56] foca em aplicar o TPC-H em bancos de dados de pequenas empresas que não podem arcar com configurações de *hardware* melhores para administrar um banco de dados. Barata, Bernardino e Furtado [57] fazem uma descrição teórica de dois *benchmarks*, o YCSB (*Yahoo Cloud Serving Benchmark*), voltado para

[1] http://www.tpc.org/information/about/abouttpc.asp
[2] Lista de sócios acessada em Junho de 2017

a área de *Big Data*, e o TPC-H. Rutishauser e Noureldin [58] realizam uma análise comparativa entre PostgreSQL e MongoDB aplicando o TPC-H. O primeiro é um SGBD relacional clássico, e o outro um SGBD NoSQL com algumas consultas do TPC-H adaptadas. Por fim, Soares [59] compara SGBD relacionais e colunares executando um teste de força sob o *Star Schema Benchmark* simulando bases de dados de 1Gb, 2Gb, 5Gb e 10Gb.

De acordo com o manual de especificação fornecido pelo TPC [60], o TPC-H é um *benchmark* de suporte à decisão de negócios, constituído de uma série de consultas comerciais *ad-hoc* e modificações simultâneas de dados com finalidade de retratar a realidade das empresas. Ele representa DSS que examinam grandes volumes de dados; executam consultas com um alto grau de complexidade; e respondem questões críticas de negócio. Como o *benchmark* trata de grandes volumes de dados, o tamanho mínimo de banco de dados proposto pelo TPC-H é de 1GB, seguido por 10GB, 30GB, 100GB, 300GB, 1000GB, 3000GB, 10000GB, 30000GB e 100000GB. Estes valores também correspondem ao Fator de Escala (*Factor Scale* – SF).

Com o intuito de avaliar o resultado do desempenho de SGBDs como DW, é necessário ter conhecimento de como os dados que irão popular o DW estão distribuídos. Para tal, o TPC-H propõe um ambiente normalizado *Snow Flake*. Ele é composto por oito tabelas e é descrito na Seção 5.2. Do mesmo modo, a fim de obedecer as regras de modelagem deste ambiente, é preciso ter dados conhecidos para popular os DWs. Estes dados são fornecidos pelo *software* DBGen.

O DBGen foi implementado pelo TPC-H, com o objetivo de realizar a população de dados em um DW seguindo a modelagem original *Snow Flake*. São gerados oito arquivos separados no formato `<nome da tabela>.tbl` – exemplificando, a tabela de dados *Part* será gerada como `part.tbl`; onde as linhas de cada arquivo representam as tuplas dentro da respectiva tabela, tendo seus atributos separados por um delimitador *pipe* ("|"). Por padrão, os arquivos são gerados para uma base de dados da classe de 1GB, quando nenhum SF é especificado.

Assim como é necessário conhecer os dados de modo a seguir a modelagem proposta pelo *benchmark*, é preciso formular consultas que visam responder às questões de negócio definidas pelo TPC-H sobre o modelo de dados. O próprio TPC define para seu modelo 22 consultas, cada qual definida por (i) uma questão de negócio, que ilustra o contexto no qual a consulta pode ser aplicada; (ii) uma definição funcional, que corresponde à implementação da consulta utilizando

a Linguagem SQL-92; (iii) parâmetros de substituição, que geram os valores necessários para completar os parâmetros da consulta; e (iv) validação, que descreve como validar a consulta no BD. Igualmente à geração de dados, a geração de consultas também é feita com o uso de um programa fornecido pelo TPC, o QGen.

Com o QGen é possível gerar as consultas de acordo com a especificação do TPC-H inserindo os parâmetros de substituição adequados em cada uma. A Consulta 1 presente no Apêndice A, Consulta de Relatório de Resumo de Preços, possui o parâmetro de substituição [DELTA], correspondente a um número inteiro entre 60 e 120, inserido ao executar o programa QGen para a respectiva consulta. Esses valores podem ser inseridos de maneira aleatória, entretanto o TPC-H define valores padrão para os parâmetros de substituição para fins de validação; portanto as consultas no QGen são geradas utilizando os valores padrão. As 22 consultas com suas respectivas questões de negócio e parâmetros de substituição são descritas no Apêndice A.

5.1 Metodologia

Após a geração dos arquivos é criado em cada SGBD um esquema correspondente às classes utilizadas, tanto para o Ambiente Normalizado quanto para o Desnormalizado. Cada SGBD é então executado sobre os dois ambientes propostos. Em cada ambiente será executado o teste de performance, que consiste de duas execuções: o Teste de Força (*Power Test*) e o Teste de Vazão (*Throughput Test*), descritos nas Subseções 5.1.1 e 5.1.2, respectivamente. O tempo resultante de cada etapa é utilizado para calcular o desempenho final do SGBD, medido em *QphH@Size* – quantidade de consultas executadas por hora, dada uma classe de tamanho de banco de dados. Uma execução é composta por:

- *SF*, que representa a classe do banco de dados.
- Q_i, que representa uma consulta, onde $1 \leqslant i \leqslant 22$.
- *S*, que define o número de sessões de consulta do Teste de Vazão.
- *s*, que representa uma dada sessão, onde $1 \leqslant s \leqslant S$.
- T_s, que representa o tempo, em segundos, resultante do processo inteiro.

- RF_j, que representa a Função de Atualização (*Refresh Function* – RF), onde:
  - RF1: inserção de novos registros. O número de registros inseridos deve ser igual para o número de registros removidos pela RF2. O pseudocódigo para a RF1 é como:

```
loop (SF * 1500) times
    insert <new row into> ORDERS table
    loop random(1, 7) times
        insert <new row into> LINEITEM table
    end loop
end loop
```

  - RF2: remoção de registros antigos. O pseudocódigo para a RF1 é como:

```
loop (SF * 1500) times
    delete from ORDERS where O_ORDERKEY = [valor]
    delete from LINEITEM where I_ORDERKEY = [valor]
end loop
```

O conjunto de dados para executar com sucesso as RFs também são gerados pelo DBGen utilizando a *flag* -U.

5.1.1 Teste de Força

O Teste de Força objetiva medir a execução de uma dada consulta do sistema com um único usuário ativo. Neste teste é criada uma única sessão com o respectivo SGBD e as seguintes instruções são executadas:

- Execução da RF1.
- Execução de cada consulta proposta pelo TPC-H, ou adaptada, de forma sequencial uma única vez até a última consulta.
- Execução da RF2.

Será armazenado ao fim do teste o tempo em segundos resultante de cada consulta, bem como o tempo de execução de cada função. Este resultado será utilizado na Equação 5.1.

$$Power@Size = \frac{3600 * SF}{\sqrt[24]{\prod_{i=2}^{i=22} Q(i,0) * \prod_{j=1}^{j=2} RF(j,0)}} \quad (5.1)$$

5.1.2 Teste de Vazão

Este teste mede a capacidade do sistema de processar a maior quantidade possível de consultas no menor intervalo de tempo. Aqui o TPC-H exige um número mínimo de sessões de consulta de acordo com a classe de tamanho do banco de dados, como mostra a Tabela 5.1.

Tabela 5.1: Número mínimo de sessões para uma classe de banco de dados

Classe do Banco de Dados	**Número de Sessões**
1 GB	2
10 GB	3
30 GB	4
100 GB	5
300 GB	6
1000 GB	7
3000 GB	8
10000 GB	9
30000 GB	10
100000 GB	11

É criada uma sessão para executar as Funções e *N* sessões para executar as consultas, de acordo com a Tabela 5.1. As instruções são executadas concorrentemente. Para a sessão das Funções, as seguintes instruções são executadas:

- A RF1 e RF2 deverão ser executadas sequencialmente *N* vezes, onde *N* é o número de sessões para a execução de consultas.
- Para cada sessão de consultas, cada consulta será executada sequencialmente até a última.

Ao fim do teste, é armazenado o tempo em segundos da execução do processo inteiro. O processo inicia quando a primeira sessão, seja de consulta ou de Função, executa sua instrução e finaliza quando a última sessão recebe uma resposta. O resultado é utilizado para calcular o desempenho do SGBD para o Teste de Vazão conforme a Equação 5.2.

$$Throughput@Size = \frac{S*22*3600}{T_s*SF} \tag{5.2}$$

O resultado dos cálculos de Força e Vazão serão utilizados para calcular o desempenho final do SGBD da seguinte forma:

$$QphH@Size=\sqrt{Power@Size*Throughput@Size} \tag{5.3}$$

5.2 Ambiente Original Normalizado

O modelo de ambiente proposto pelo TPC-H é um esquema normalizado *Snow Flake*. Ele é composto por oito tabelas, sendo que destas, seis têm o tamanho multiplicado por um Fator de Escala, que corresponde ao tamanho da classe do banco de dados; enquanto que as demais, *NATION* e *REGION*, têm tamanho fixo. O relacionamento *one-to-many* entre as tabelas do esquema *Snow Flake* é apresentado na Figura 5.1, adaptada do manual de especificação do TPC-H [60].

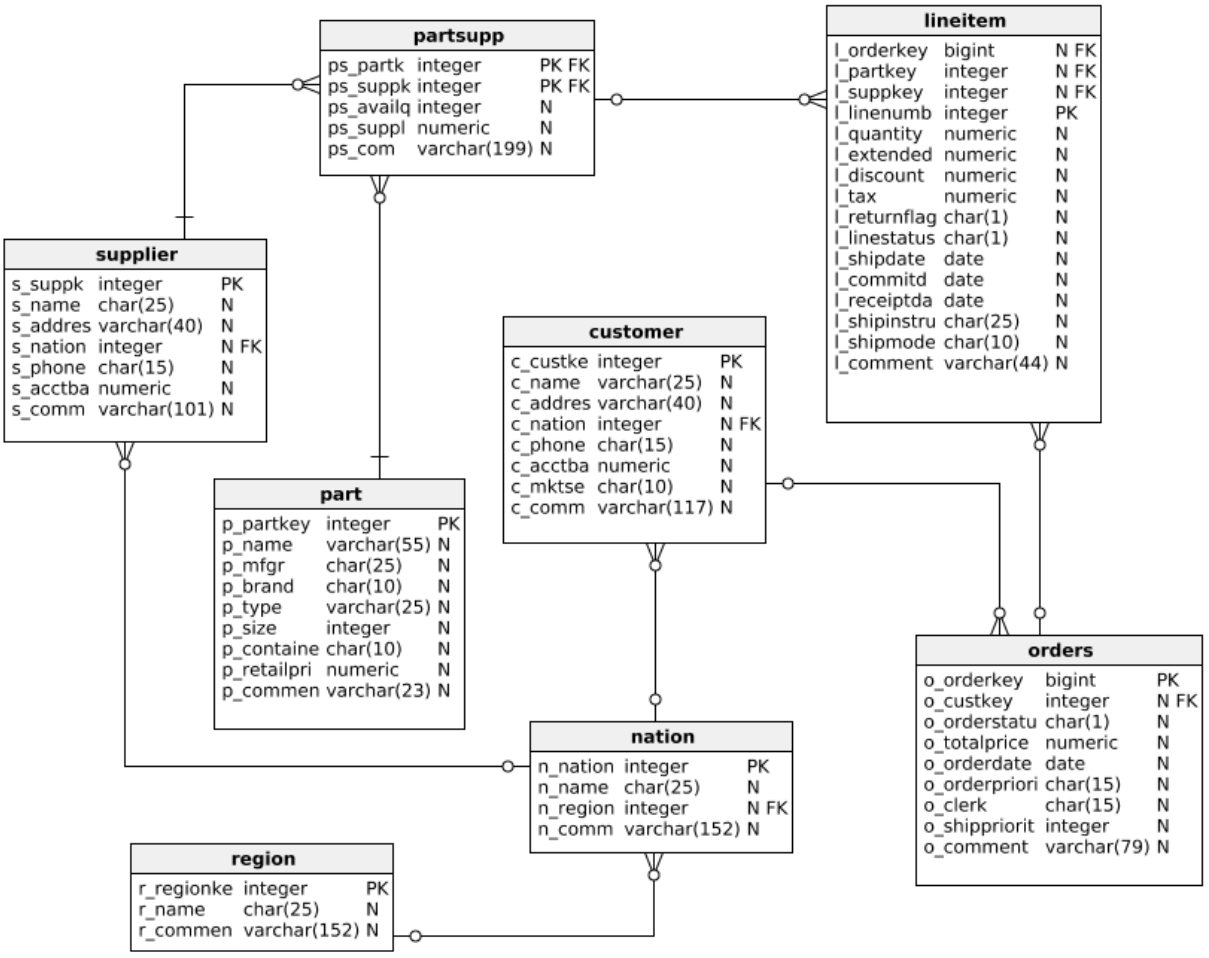

Figura 5.1: Esquema do ambiente normalizado

Os tipos de dados atribuídos aos atributos de uma entidade podem ser:

- Identificador: *identifier* – corresponde a um inteiro que define a PK de uma entidade
- Inteiro: int

- Decimal: decimal
- Texto fixo de tamanho N: text(N)
- Texto variável de tamanho N: varchar(N)
- Data: date

Detalhes sobre as consultas implementadas na Linguagem SQL relacionadas a esse ambiente são encontradas no Apêndice A.

5.3 Ambiente Desnormalizado

Embora modelos de dados normalizados sejam eficientes em processos operacionais, SGBDs relacionais não executam uma consulta de maneira eficiente em um esquema normalizado [4]. A otimização é afetada pelo número de *joins*, o que vai contra um dos objetivos de um *warehouse*: recuperação de dados com rapidez.

As modificações no modelo do Ambiente Normalizado se fundamentaram na criação de um modelo parecido com o apresentado pela modelagem *Star*, com uma Tabela Fato central descrita por Tabelas Dimensão. Também procurou-se eliminar quaisquer tabelas cujos atributos sejam frequentemente utilizados em operações de *join*, alguns característicos de um modelo com Tabelas Subdimensionais, e que possam ser incluídos nas tabelas que possuam relacionamento com eles. O esquema final do modelo *Star* é mostrado na Figura 5.2.

A primeira modificação no modelo *Snow Flake* foi a exclusão das entidades Dimensão *Nation* e *Region*. Essas entidades são Tabelas Subdimensionais das entidades *Customer* e *Supplier*, sendo frequentemente requeridas quando há a necessidade de consultar a nação e/ou a região de um cliente e/ou de um fornecedor, assim optou-se por incluir os atributos `N_NAME`, `N_COMMENT`, `R_NAME` e `R_COMMENT` nas entidades *Supplier* e *Customer*.

Com a intenção de criar uma única Tabela Fato no esquema desnormalizado, observou-se (i) a possibilidade de unir as entidades *Lineitem* e *Orders* em uma única entidade nomeada como *Item*, visto que as duas possuem um relacionamento. Isto também eliminaria alguns *joins* realizados entre as duas entidades nas consultas listadas no Apêndice A. A PK `C_CUSTKEY` de *Customer* que antes estava em *Orders* como FK é transferida agora para *Item*. Também

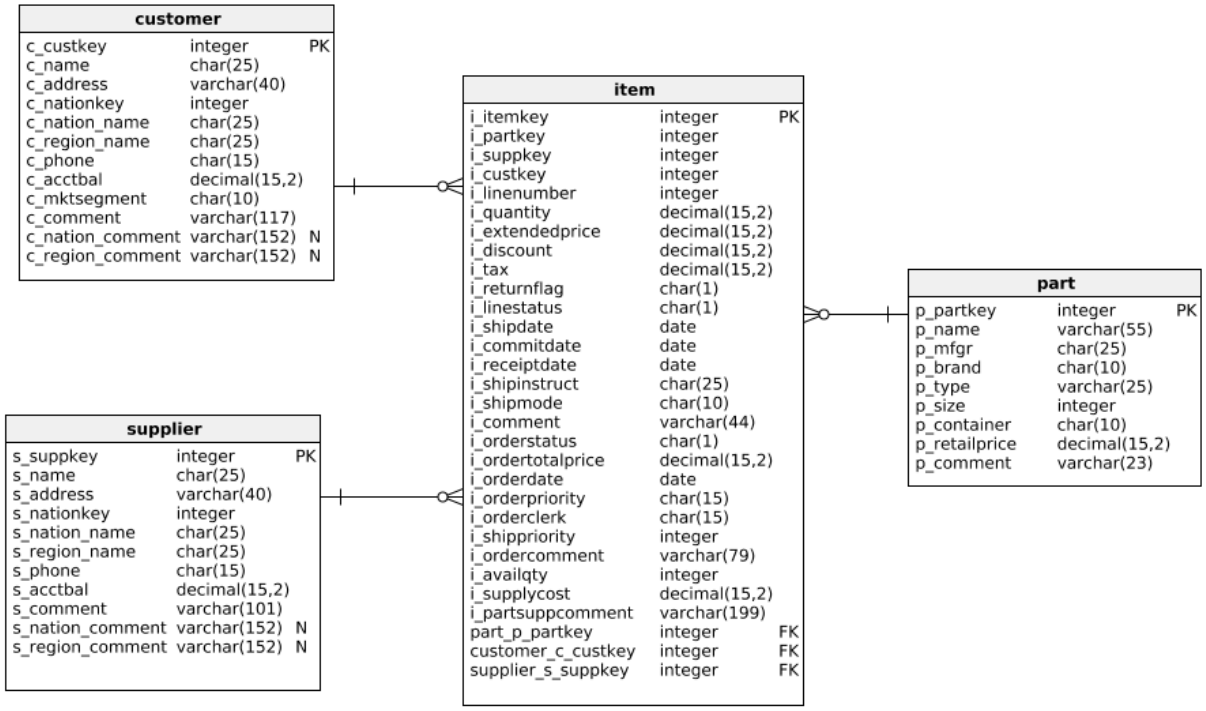

Figura 5.2: Esquema do ambiente desnormalizado

nota-se que (ii) as entidades *Part* e *Supplier* têm suas PKs como FKs na entidade antes denominada *Lineitem*, provenientes da entidade intermediária *Partsupp*. Optou-se assim por incluir os atributos de *Partsupp* na nova entidade *Item*, excluindo a primeira do esquema e mantendo o relacionamento de *Supplier* e *Part* diretamente com *Item*, através das chaves `P_PARTKEY` e `S_SUPPKEY`.

O esquema final agora possui uma Tabela Fato, *Item*; e três tabelas Dimensão, *Part*, *Customer* e *Supplier*, cada qual com sua chave mantendo o relacionamento com a tabela *Item*. Os dados presentes nas entidades são do mesmo tipo descritos na Seção 5.2. As consultas em Linguagem SQL relacionadas a esse ambiente são encontradas no Apêndice B.

A inserção dos dados do Ambiente Desnormalizado será realizada após todos os dados gerados pelo DBGen do Ambiente Normalizado terem sido populados nos SGBDs. Com isso, é possível realizar *joins* de acordo com as mudanças e o resultado destes será armazenado no esquema correspondente. Por exemplo, ao realizar um *join* entre as entidades *Nation* e *Region* selecionando todos os seus atributos, estes são enviados como entrada para a inserção em uma nova tabela do esquema Desnormalizado correspondente, digamos, *Nation_Region*. Após, é realizado um novo *join* com as entidades *Supplier* e *Part*, separadamente, para incluir os atri-

butos de *Nation_Region*: `N_NAME, N_COMMENT, R_NAME` e `R_COMMENT`; bem como os atributos originais das duas entidades anteriores, nas novas entidades *Supplier* e *Part*.

Capítulo 6

O Experimento

Como mencionado no Capítulo 3, muitas empresas ainda utilizam bancos relacionais como forma de gerenciar seus dados. Existem vários SGBDR atuantes no mercado, como MySQL, PostgreSQL e algumas variantes da Oracle. Por se tratar de um SGBD *open source* e ser amplamente conhecido, foi escolhido para o estudo o PostgreSQL. Ele possui vasta documentação, tutoriais e suporte em fóruns sobre bancos de dados e programação de forma geral, além de receber recorrentes atualizações.

A categoria de NoSQL escolhida foi a colunar, por ser a mais recomendada para gerenciamento do grande volume de dados de um DW. O fato do PostgreSQL utilizar a Linguagem SQL como forma de manipulação de dados faz com que seja necessário também escolher o banco colunar para comparação utilizando esse filtro. A escolha também se deu com base na simplicidade do uso do SGBD, e na praticidade de migração dos dados armazenados entre um banco e outro. Para tanto, o MonetDB foi definido para o estudo.

O gerenciamento dos SGBD foi realizado através do *driver* JDBC disponibilizados nas páginas oficiais do PostgreSQL e do MonetDB. Seu uso se deu devido à simplificação no desenvolvimento de aplicações, por dar suporte a diferentes SGBD e possuir boa documentação e tutoriais. Na análise de tempo foi considerada a latência da Máquina Virtual Java (JVM), assim incluindo essa latência no tempo total. As versões dos SGBD utilizadas foram o PostgreSQL 9.6 e o MonetDB XXX e foram instalados no servidor do laboratório GIA, operando o sistema operacional Ubuntu Linux 16.04 LTS, com 16Gb de RAM e dois processadores Intel Xeon CPU E5620 2.40GHz.

Os cenários de teste consistiram de três bases de dados de tamanhos diferentes, de 1Gb, 10Gb e 30Gb para simular desde uma base pequena até bases maiores. Cada cenário foi execu-

tado nos ambientes *snowflake* e *star*.

Alguns fatores devem ser considerados ao executar testes de *benchmark* no escopo de banco de dados [61]:

1. O ambiente de execução deve ser o mesmo para todos os SGBD, bem como a lógica e sequência de execução.

2. A modelagem do banco deve ser idêntica para todos os modelos, e isso inclui os tipos de dados e restrições de integridade. Ao analisar os resultados do *benchmark*, a menos que seja especificado no estudo, não há como saber qual é o tipo de dado de um atributo quando existem diferentes tipos que podem ser usados para a mesma função, e essa diferença acarreta em resultados diferentes de desempenho. Por exemplo, em Raasveldt et al. [61] o SGBD MariaDB apresentou diferenças quando implementado utilizando DECIMAL e DOUBLE.

3. *Cold run* vs. *Hot run*: em alguns sistemas a primeira execução de testes (*cold*) costuma levar mais tempo para executar que as execuções subsequentes (*hot*). Isso se dá pelo buffer do banco de dados ou o próprio sistema operacional ter armazenado em cache os dados requeridos.

Os itens 1 e 2 foram tratados antes da execução do TPC-H. Nos Apêndices C e D se encontram as SQL utilizadas para criar os esquemas *snowflake* e *star* nos SGBD. O fluxograma da Figura A ilustra ainda a sequência de passos desde a criação dos bancos até a execução do *benchmark*.

O item 3 foi tratado durante a execução das consultas do teste de força: para cada cenário é realizada três análises diferentes, a primeira considerando a primeira execução de consultas; a segunda considerando o melhor resultado entre três execuções; e a terceira limpando parte da cache do sistema operacional antes de rodar a terceira execução. Essa análise é feita somente sob as consultas do teste de força por ser o primeiro teste a ser feito, como pode ser observado na Figura A. No teste de vazão todas as consultas já foram executadas uma vez e as funções de atualização trabalham sobre dados diferentes a cada vez que são executadas, impossibilitando o armazenamento em cache.

Para averiguar se todas as 22 consultas do TPC-H executam de forma correta e em tempo hábil, principalmente atentando-se ao fato de que no ambiente denormalizado elas foram modificadas, foi realizada uma execução prévia de cada uma, sob a menor base de dados, utilizando o JDBC. Dessa execução, foram eliminadas as consultas Q1, Q2, Q4, Q15, Q17, Q20 e Q21 por terem demorado horas no ambiente denormalizado – e algumas tendo a execução cancelada antes – enquanto que as demais levaram apenas alguns segundos para executar.

6.1 Carregamento de Dados

6.2 Base de 1GB

6.3 Base de 10GB

6.4 Base de 30GB

Capítulo 7

Conclusão

Apêndice A

Consultas do Ambiente Original Normalizado

São apresentadas aqui as 15 consultas realizadas no Ambiente *snowflake*, cada qual com sua questão de negócio brevemente explicada.

1. Consulta de Prioridade de Envio

   Retorna a prioridade de envio dos pedidos com a maior receita entre aqueles que ainda não foram enviados em uma determinada data.

Parâmetro de Substituição	Valor
SEGMENT	BUILDING
DATE	1995-03-15

```
select
    l_orderkey,
    sum(l_extendedprice * (1 - l_discount)) as revenue,
    o_orderdate,
    o_shippriority
from
    customer,
    orders,
    lineitem
where
    c_mktsegment = '[SEGMENT]'
    and c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and o_orderdate < date '[DATE]'
    and l_shipdate > date '[DATE]'
group by
    l_orderkey,
    o_orderdate,
    o_shippriority
```

```
order by
   revenue desc,
   o_orderdate;
set rowcount 10
go
```

2. Consulta de Volume do Fornecedor Local

   Lista para cada nação em uma dada região o volume de receita originado de uma transação na qual cliente e fornecedor eram daquela nação.

Parâmetro de Substituição	Valor
REGION	ASIA
DATE	1994-01-01

```
select
   n_name,
   sum(l_extendedprice * (1 - l_discount)) as revenue
from
   customer,
   orders,
   lineitem,
   supplier,
   nation,
   region
where
   c_custkey = o_custkey
   and l_orderkey = o_orderkey
   and l_suppkey = s_suppkey
   and c_nationkey = s_nationkey
   and s_nationkey = n_nationkey
   and n_regionkey = r_regionkey
   and r_name = '[REGION]'
   and o_orderdate >= date '[DATE]'
   and o_orderdate < date '[DATE]' + interval '1' year
group by
   n_name
order by
   revenue desc;
```

3. Previsão de Mudança de Receita

   Informa o quanto a receita pode aumentar eliminando alguns descontos em um dado ano.

Parâmetro de Substituição	Valor
DATE	1994-01-01
DISCOUNT	.06
QUANTITY	24

```
select
   sum(l_extendedprice * l_discount) as revenue
from
   lineitem
where
   l_shipdate >= date '[DATE]'
   and l_shipdate < date '[DATE]' + interval '1' year
   and l_discount between [DISCOUNT] - 0.01 and [DISCOUNT] + ↩
      0.01
   and l_quantity < [QUANTITY];
```

4. Consulta de Quantidade de Envio

   Determina o valor de bens enviados entre nações para auxiliar na renegociação de contratos de envio.

Parâmetro de Substituição	Valor
NATION1	FRANCE
NATION2	GERMANY

```
select
   supp_nation,
   cust_nation,
   l_year,
   sum(volume) as revenue
from
   (
      select
         n1.n_name as supp_nation,
         n2.n_name as cust_nation,
         extract(year from l_shipdate) as l_year,
         l_extendedprice * (1 - l_discount) as volume
      from
         supplier,
         lineitem,
         orders,
         customer,
         nation n1,
         nation n2
      where
         s_suppkey = l_suppkey
```

```
            and o_orderkey = l_orderkey
            and c_custkey = o_custkey
            and s_nationkey = n1.n_nationkey
            and c_nationkey = n2.n_nationkey
            and (
               (n1.n_name = '[NATION1]' and n2.n_name = '[NATION2]↩
                  ')
               or (n1.n_name = '[NATION2]' and n2.n_name = '[↩
                  NATION1]')
            )
            and l_shipdate between date '1995-01-01' and date '↩
               1996-12-31'
   ) as shipping
group by
   supp_nation,
   cust_nation,
   l_year
order by
   supp_nation,
   cust_nation,
   l_year;
```

5. Consulta de Quota de Mercado Nacional

Determina quanto a quota de mercado de uma dada nação em uma dada região mudou em dois anos para um determinado tipo de peça.

Parâmetro de Substituição	Valor
NATION	BRAZIL
REGION	AMERICA
TYPE	ECONOMY ANODIZED STEEL

```
select
   o_year,
   sum(case
      when nation = '[NATION]'
      then volume
      else 0
   end) / sum(volume) as mkt_share
from
   (
      select
         extract(year from o_orderdate) as o_year,
         l_extendedprice * (1 - l_discount) as volume,
         n2.n_name as nation
      from
         part,
```

```
            supplier,
            lineitem,
            orders,
            customer,
            nation n1,
            nation n2,
            region
        where
            p_partkey = l_partkey
            and s_suppkey = l_suppkey
            and l_orderkey = o_orderkey
            and o_custkey = c_custkey
            and c_nationkey = n1.n_nationkey
            and n1.n_regionkey = r_regionkey
            and r_name = '[REGION]'
            and s_nationkey = n2.n_nationkey
            and o_orderdate between date '1995-01-01' and date '↩
                1996-12-31'
                    and p_type = '[TYPE]'
    ) as all_nations
group by
    o_year
order by
    o_year;
```

6. Consulta ao Lucro de um Tipo de Produto

Encontra para cada nação em cada ano o lucro de todas as peças pedidas naquele ano contendo uma substring específica em seus nomes que foram preenchidos por um fornecedor naquela nação.

Parâmetro de Substituição	Valor
COLOR	green

```
select
    nation,
    o_year,
    sum(amount) as sum_profit
from
    (
        select
            n_name as nation,
            extract(year from o_orderdate) as o_year,
            l_extendedprice * (1 - l_discount) - ps_supplycost * ↩
                l_quantity as a
mount
```

```
        from
            part,
            supplier,
            lineitem,
            partsupp,
            orders,
            nation
        where
            s_suppkey = l_suppkey
            and ps_suppkey = l_suppkey
            and ps_partkey = l_partkey
            and p_partkey = l_partkey
            and o_orderkey = l_orderkey
            and s_nationkey = n_nationkey
            and p_name like '%[COLOR]%'
    ) as profit
group by
    nation,
    o_year
order by
    nation,
    o_year desc;
```

7. Consulta de Relatório de Itens Retornados

   Retorna os 20 principais clientes que podem estar tendo problemas com peças que foram enviadas a eles em um dado trimestre.

Parâmetro de Substituição	Valor
DATE	1993-10-01

```
select
    c_custkey,
    c_name,
    sum(l_extendedprice * (1 - l_discount)) as revenue,
    c_acctbal,
    n_name,
    c_address,
    c_phone,
    c_comment
from
    customer,
    orders,
    lineitem,
    nation
where
    c_custkey = o_custkey
```

```
    and l_orderkey = o_orderkey
    and o_orderdate >= date '[DATE]'
    and o_orderdate < date '[DATE]' + interval '3' month
    and l_returnflag = 'R'
    and c_nationkey = n_nationkey
group by
    c_custkey,
    c_name,
    c_acctbal,
    c_phone,
    n_name,
    c_address,
    c_comment
order by
    revenue desc;
set rowcount 20
go
```

8. Consulta a Identificação de Estoques Importantes

Avalia de todos os estoques de fornecedores disponíveis em uma dada nação as peças com uma percentagem significativa do total de peças disponíveis.

Parâmetro de Substituição	Valor
NATION	GERMANY
FRACTION	.0001

```
select
    ps_partkey,
    sum(ps_supplycost * ps_availqty) as value
from
    partsupp,
    supplier,
    nation
where
    ps_suppkey = s_suppkey
    and s_nationkey = n_nationkey
    and n_name = '[NATION]'
group by
    ps_partkey having
        sum(ps_supplycost * ps_availqty) > (
            select
                sum(ps_supplycost * ps_availqty) * [FRACTION]
            from
                partsupp,
                supplier,
                nation
```

```
            where
                ps_suppkey = s_suppkey
                and s_nationkey = n_nationkey
                and n_name = '[NATION]'
        )
order by
    value desc;
```

9. Consulta a Modos de Envio e Prioridade de Pedidos

   Determina quando selecionar modos de envio mais baratos afeta negativamente pedidos com alta prioridade.

Parâmetro de Substituição	Valor
SHIPMODE1	MAIL
SHIPMODE2	SHIP
DATE	1994-01-01

```
select
    l_shipmode,
    sum(case
        when o_orderpriority = '1-URGENT'
            or o_orderpriority = '2-HIGH'
            then 1
        else 0
    end) as high_line_count,
    sum(case
        when o_orderpriority <> '1-URGENT'
            and o_orderpriority <> '2-HIGH'
            then 1
        else 0
    end) as low_line_count
from
    orders,
    lineitem
where
    o_orderkey = l_orderkey
    and l_shipmode in ('[SHIPMODE1]', '[SHIPMODE2]')
    and l_commitdate < l_receiptdate
    and l_shipdate < l_commitdate
    and l_receiptdate >= date '[DATE]'
    and l_receiptdate < date '[DATE]' + interval '1' year
group by
    l_shipmode
order by
    l_shipmode;
```

10. Consulta à Distribuição de Clientes

Determina a distribuição de clientes pelo número de pedidos que eles fizeram.

Parâmetro de Substituição	Valor
WORD1	special
WORD2	requests

```
select
   c_count,
   count(*) as custdist
from
   (
      select
         c_custkey,
         count(o_orderkey)
      from
         customer left outer join orders on
            c_custkey = o_custkey
            and o_comment not like '%[WORD1]%[WORD2]%'
      group by
         c_custkey
   ) as c_orders (c_custkey, c_count)
group by
   c_count
order by
   custdist desc,
   c_count desc;
```

11. Consulta aos Efeitos de Promoção

Informa o retorno de mercado a uma propaganda, como um comercial de televisão ou uma campanha especial.

Parâmetro de Substituição	Valor
DATE	1995-09-01

```
select
   100.00 * sum(case
      when p_type like 'PROMO%'
         then l_extendedprice * (1 - l_discount)
      else 0
   end) / sum(l_extendedprice * (1 - l_discount)) as ↩
      promo_revenue
from
   lineitem,
   part
```

```
where
   l_partkey = p_partkey
   and l_shipdate >= date '[DATE]'
   and l_shipdate < date '[DATE]' + interval '1' month;
```

12. Consulta à Relação *Part/Supplier*

Descobre quantos fornecedores podem fornecer peças com determinados atributos requeridos por um cliente.

Parâmetro de Substituição	Valor
BRAND	Brand#45
TYPE	MEDIUM POLISHED
SIZE1	49
SIZE2	14
SIZE3	23
SIZE4	45
SIZE5	19
SIZE6	3
SIZE7	36
SIZE8	9

```
select
   p_brand,
   p_type,
   p_size,
   count(distinct ps_suppkey) as supplier_cnt
from
   partsupp,
   part
where
   p_partkey = ps_partkey
   and p_brand <> '[BRAND]'
   and p_type not like '[TYPE]%'
   and p_size in ([SIZE1], [SIZE2], [SIZE3], [SIZE4], [SIZE5], ↩
      [SIZE6], [SIZE7], [SIZE8])
   and ps_suppkey not in (
      select
         s_suppkey
      from
         supplier
      where
         s_comment like '%Customer%Complaints%'
   )
group by
   p_brand,
```

```
    p_type,
    p_size
order by
    supplier_cnt desc,
    p_brand,
    p_type,
    p_size;
```

13. Consulta a Grandes Volumes de Clientes

Classifica os 100 principais clientes que já realizaram grandes quantidades de pedidos.

Parâmetro de Substituição	Valor
QUANTITY	300

```
select
    c_name,
    c_custkey,
    o_orderkey,
    o_orderdate,
    o_totalprice,
    sum(l_quantity)
from
    customer,
    orders,
    lineitem
where
    o_orderkey in (
        select
            l_orderkey
        from
            lineitem
        group by
            l_orderkey having
                sum(l_quantity) > [QUANTITY]
    )
    and c_custkey = o_custkey
    and o_orderkey = l_orderkey
group by
    c_name,
    c_custkey,
    o_orderkey,
    o_orderdate,
    o_totalprice
order by
    o_totalprice desc,
    o_orderdate;
```

14. Consulta a Desconto de Receita

Encontra o desconto bruto de todos os pedidos para três diferentes tipos de peças que foram enviadas por via aérea e entregues pessoalmente.

Parâmetro de Substituição	Valor
QUANTITY1	300
BRAND1	Brand#12
QUANTITY2	10
BRAND2	Brand#23
QUANTITY3	20
BRAND3	Brand#34

```
select
   sum(l_extendedprice* (1 - l_discount)) as revenue
from
   lineitem,
   part
where
   (
      p_partkey = l_partkey
      and p_brand = '[BRAND1]'
      and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM ↩
         PKG')
      and l_quantity >= [QUANTITY1] and l_quantity <= [↩
         QUANTITY1] + 10
      and p_size between 1 and 5
      and l_shipmode in ('AIR', 'AIR REG')
      and l_shipinstruct = 'DELIVER IN PERSON'
   )
   or
   (
      p_partkey = l_partkey
      and p_brand = '[BRAND2]'
      and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED↩
          PACK')
      and l_quantity >= [QUANTITY2] and l_quantity <= [↩
         QUANTITY2] + 10
      and p_size between 1 and 10
      and l_shipmode in ('AIR', 'AIR REG')
      and l_shipinstruct = 'DELIVER IN PERSON'
   )
   or
   (
      p_partkey = l_partkey
      and p_brand = '[BRAND3]'
      and p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG ↩
```

```
        PKG')
    and l_quantity >= [QUANTITY3] and l_quantity <= [↩
        QUANTITY3] + 10
    and p_size between 1 and 15
    and l_shipmode in ('AIR', 'AIR REG')
    and l_shipinstruct = 'DELIVER IN PERSON'
);
```

15. Consulta a Oportunidades de Vendas Globais

Conta quantos clientes de determinados países não realizaram pedidos durante sete anos, porém têm chances de realizar um pedido.

Parâmetro de Substituição	Valor
I1	13
I2	31
I3	23
I4	29
I5	30
I6	18
I7	17

```
select
    cntrycode,
    count(*) as numcust,
    sum(c_acctbal) as totacctbal
from
    (
        select
            substring(c_phone from 1 for 2) as cntrycode,
            c_acctbal
        from
            customer
        where
            substring(c_phone from 1 for 2) in
                ('[I1]', '[I2]', '[I3]', '[I4]', '[I5]', '[I6]', '[↩
                    I7]')
            and c_acctbal > (
                select
                    avg(c_acctbal)
                from
                    customer
                where
                    c_acctbal > 0.00
                    and substring(c_phone from 1 for 2) in
                        ('[I1]', '[I2]', '[I3]', '[I4]', '[I5]', '[I6↩
                            ]', '[I7]')
```

```
        )
        and not exists (
            select
                *
            from
                orders
            where
                o_custkey = c_custkey
        )
    ) as custsale
group by
    cntrycode
order by
    cntrycode;
```

Apêndice B

Consultas do Ambiente Denormalizado

São apresentadas aqui as 15 consultas realizadas no Ambiente Denormalizado adaptado da proposta original do TPC-H. As questões de negócio são as mesmas que as apresentadas no Apêndice A.

1. Consulta de Prioridade de Envio

Parâmetro de Substituição	Valor
SEGMENT	BUILDING
DATE	1995-03-15

```
select
   i_itemkey,
   sum(i_extendedprice * (1 - i_discount)) as revenue,
   i_order_orderdate,
   i_order_shippriority
from
   customer,
   item
where
   c_mktsegment = '[SEGMENT]'
   and c_custkey = i_custkey
   and i_order_orderdate < date '[DATE]'
   and i_shipdate > date '[DATE]'
group by
   i_itemkey,
   i_order_orderdate,
   i_order_shippriority
order by
   revenue desc,
   o_orderdate
set rowcount 10
go
```

2. Consulta de Volume do Fornecedor Local

Parâmetro de Substituição	Valor
REGION	ASIA
DATE	1994-01-01

```
select
   s_nation_name,
   sum(i_extendedprice * (1 - i_discount)) as revenue
from
   customer,
   item
   supplier,
where
   c_custkey = i_custkey
   and i_suppkey = s_suppkey
   and c_nation_name = s_nation_name
   and s_region_name = '[REGION]'
   and i_order_orderdate >= date '[DATE]'
   and i_order_orderdate < date '[DATE]' + interval '1' year
group by
   s_nation_name
order by
   revenue desc;
```

3. Previsão de Mudança de Receita

Parâmetro de Substituição	Valor
DATE	1994-01-01
DISCOUNT	.06
QUANTITY	24

```
select
   sum(i_extendedprice * i_discount) as revenue
from
   item
where
   i_shipdate >= date '[DATE]'
   and i_shipdate < date '[DATE]' + interval '1' year
   and i_discount between [DISCOUNT] - 0.01 and [DISCOUNT] + ↩
      0.01
   and i_quantity < [QUANTITY];
```

4. Consulta de Quantidade de Envio

Parâmetro de Substituição	Valor
NATION1	FRANCE
NATION2	GERMANY

```
select
   supp_nation,
   cust_nation,
   i_year,
   sum(volume) as revenue
from
   (
      select
         s_nation_name as supp_nation,
         c_nation_name as cust_nation,
         extract(year from i_shipdate) as i_year,
         i_extendedprice * (1 - i_discount) as volume
      from
         supplier,
         item,
         customer,
      where
         s_suppkey = i_suppkey
         and c_custkey = i_custkey
         and (
            (s_nation_name = '[NATION1]' and c_nation_name = '[↩
               NATION2]')
            or (s_nation_name = '[NATION2]' and c_nation_name =↩
                '[NATION1]')
         )
         and i_shipdate between date '1995-01-01' and date '↩
            1996-12-31'
   ) as shipping
group by
   supp_nation,
   cust_nation,
   i_year
order by
   supp_nation,
   cust_nation,
   i_year;
```

5. Consulta de Quota de Mercado Nacional

Parâmetro de Substituição	Valor
NATION	BRAZIL
REGION	AMERICA
TYPE	ECONOMY ANODIZED STEEL

```
select
   order_year,
   sum(case
      when nation = '[NATION]' then volume
```

```
        else 0
    end) / sum(volume) as mkt_share
from
    (
        select
            extract(year from i_order_orderdate) as order_year,
            i_extendedprice * (1 - i_discount) as volume,
            s_nation_name as nation
        from
            part,
            supplier,
            item,
            customer
        where
            p_partkey = i_partkey
            and s_suppkey = i_suppkey
            and i_custkey = c_custkey
            and c_region_name = '[REGION]'
            and i_order_orderdate between date '1995-01-01' and ↩
                date '1996-12-31'
            and p_type = '[TYPE]'
    ) as all_nations
group by
    order_year
order by
    order_year;
```

6. Consulta ao Lucro de um Tipo de Produto

Parâmetro de Substituição	Valor
COLOR	green

```
select
    nation,
    order_year,
    sum(amount) as sum_profit
from
    (
        select
            s_nation_name as nation,
            extract(year from i_order_orderdate) as order_year,
            i_extendedprice * (1 - i_discount) - ↩
                i_partsupp_supplycost * i_quantity as amount
        from
            part,
            supplier,
            item
        where
```

```
            s_suppkey = i_suppkey
            and p_partkey = i_partkey
            and p_name like '%[COLOR]%'
    ) as profit
group by
    nation,
    order_year
order by
    nation,
    order_year desc;
```

7. Consulta de Relatório de Itens Retornados

Parâmetro de Substituição	Valor
DATE	1993-10-01

```
select
    c_custkey,
    c_name,
    sum(i_extendedprice * (1 - i_discount)) as revenue,
    c_acctbal,
    c_nation_name,
    c_address,
    c_phone,
    c_comment
from
    customer,
    item
where
    c_custkey = i_custkey
    and i_order_orderdate >= date '[DATE]'
    and i_order_orderdate < date '[DATE]' + interval '3' month
    and i_returnflag = 'R'
group by
    c_custkey,
    c_name,
    c_acctbal,
    c_phone,
    c_nation_name,
    c_address,
    c_comment
order by
    revenue desc;

set rowcount 20
go
```

8. Consulta a Identificação de Estoques Importantes

Parâmetro de Substituição	Valor
NATION	GERMANY
FRACTION	.0001

```
select
   i_partkey,
   sum(i_partsupp_supplycost * i_partsupp_availqty) as value
from
   item,
   supplier
where
   i_suppkey = s_suppkey
   and s_nation_name = '[NATION]'
group by
   i_partkey having
      sum(i_partsupp_supplycost * i_partsupp_availqty) > (
         select
            sum(i_partsupp_supplycost * i_partsupp_availqty) * ↩
               [FRACTION]
         from
            item,
            supplier
         where
            i_suppkey = s_suppkey
            and s_nation_name = '[NATION]'
      )
order by
   value desc;
```

9. Consulta a Modos de Envio e Prioridade de Pedidos

Parâmetro de Substituição	Valor
SHIPMODE1	MAIL
SHIPMODE2	SHIP
DATE	1994-01-01

```
select
   i_shipmode,
   sum(case
      when i_order_orderpriority = '1-URGENT'
         or i_order_orderpriority = '2-HIGH'
         then 1
      else 0
   end) as high_line_count,
   sum(case
```

```
        when i_order_orderpriority <> '1-URGENT'
           and i_order_orderpriority <> '2-HIGH'
           then 1
        else 0
    end) as low_line_count
from
    item
where
    and i_shipmode in ('[SHIPMODE1]', '[SHIPMODE2]')
    and i_commitdate < i_receiptdate
    and i_shipdate < i_commitdate
    and i_receiptdate >= date '[DATE]'
    and i_receiptdate < date '[DATE]' + interval '1' year
group by
    i_shipmode
order by
    i_shipmode;
```

10. Consulta à Distribuição de Clientes

Parâmetro de Substituição	Valor
WORD1	special
WORD2	requests

```
select
    c_count,
    count(*) as custdist
from
    (
        select
            c_custkey,
            count(i_itemkey)
        from
            customer left outer join item on
                c_custkey = i_custkey
                and i_order_comment not like '%[WORD1]%[WORD2]%↩
                    '
        group by
                c_custkey
    ) as c_orders (c_custkey, c_count)
group by
    c_count
order by
    custdist desc,
    c_count desc;
```

11. Consulta aos Efeitos de Promoção

Parâmetro de Substituição	Valor
DATE	1995-09-01

```
select
   100.00 * sum(case
      when p_type like 'PROMO%'
         then i_extendedprice * (1 - i_discount)
      else 0
   end) / sum(i_extendedprice * (1 - i_discount)) as ↩
      promo_revenue
from
   item,
   part
where
   i_partkey = p_partkey
   and i_shipdate >= date '[DATE]'
   and i_shipdate < date '[DATE]' + interval '1' month;
```

12. Consulta à Relação *Part/Supplier*

Parâmetro de Substituição	Valor
BRAND	Brand#45
TYPE	MEDIUM POLISHED
SIZE1	49
SIZE2	14
SIZE3	23
SIZE4	45
SIZE5	19
SIZE6	3
SIZE7	36
SIZE8	9

```
select
    p_brand,
    p_type,
    p_size,
    count(distinct i_suppkey) as supplier_cnt
from
   item,
    part
where
    p_partkey = i_partkey
    and p_brand <> '[BRAND]'
    and p_type not like '[TYPE]%'
    and p_size in ([SIZE1], [SIZE2], [SIZE3], [SIZE4], [SIZE5],↩
        [SIZE6], [SIZE7], [SIZE8])
```

```
    and i_suppkey not in (
        select
            s_suppkey
        from
            supplier
        where
            s_comment like '%Customer%Complaints%'
    )
group by
    p_brand,
    p_type,
    p_size
order by
    supplier_cnt desc,
    p_brand,
    p_type,
    p_size;
go
```

13. Consulta a Grandes Volumes de Clientes

Parâmetro de Substituição	Valor
QUANTITY	300

```
select
    c_name,
    c_custkey,
    i_itemkey,
    i_order_orderdate,
    i_order_totalprice,
    sum(i_quantity)
from
    customer,
    item
where
    i_itemkey in (
        select
            i_itemkey
        from
            item
        group by
            i_itemkey having
                sum(i_quantity) > [QUANTITY]
    )
    and c_custkey = i_custkey
group by
    c_name,
    c_custkey,
```

```
    i_itemkey,
    i_order_orderdate,
    i_order_totalprice
order by
    i_order_totalprice desc,
    i_order_orderdate;
set rowcount 100
go
```

14. Consulta a Desconto de Receita

Parâmetro de Substituição	Valor
QUANTITY1	300
BRAND1	Brand#12
QUANTITY2	10
BRAND2	Brand#23
QUANTITY3	20
BRAND3	Brand#34

```
select
    sum(i_extendedprice* (1 - i_discount)) as revenue
from
    item,
    part
where
    (
        p_partkey = i_partkey
        and p_brand = '[BRAND1]'
        and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM↩
            PKG')
        and i_quantity >= [QUANTITY1] and i_quantity <= [↩
           QUANTITY1] + 10
        and p_size between 1 and 5
        and i_shipmode in ('AIR', 'AIR REG')
        and i_shipinstruct = 'DELIVER IN PERSON'
    )
    or
    (
        p_partkey = i_partkey
        and p_brand = '[BRAND2]'
        and p_container in ('MED BAG', 'MED BOX', 'MED PKG', '↩
           MED PACK')
        and i_quantity >= [QUANTITY2] and i_quantity <= [↩
           QUANTITY2] + 10
        and p_size between 1 and 10
        and i_shipmode in ('AIR', 'AIR REG')
        and i_shipinstruct = 'DELIVER IN PERSON'
    )
```

```
    or
    (
        p_partkey = i_partkey
        and p_brand = '[BRAND3]'
        and p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG↩
            PKG')
        and i_quantity >= [QUANTITY3] and i_quantity <= [↩
           QUANTITY3] + 10
        and p_size between 1 and 15
        and i_shipmode in ('AIR', 'AIR REG')
        and i_shipinstruct = 'DELIVER IN PERSON'
    );
```

15. Consulta a Oportunidades de Vendas Globais

Parâmetro de Substituição	Valor
I1	13
I2	31
I3	23
I4	29
I5	30
I6	18
I7	17

```
select
    cntrycode,
    count(*) as numcust,
    sum(c_acctbal) as totacctbal
from
    (
        select
            substring(c_phone from 1 for 2) as cntrycode,
            c_acctbal
        from
            customer
        where
            substring(c_phone from 1 for 2) in
                ('[I1]', '[I2]', '[I3]', '[I4]', '[I5]', '[I6]'↩
                    , '[I7]')
            and c_acctbal > (
                select
                    avg(c_acctbal)
                from
                    customer
                where
                    c_acctbal > 0.00
                    and substring(c_phone from 1 for 2) in
```

```
                                ('[I1]', '[I2]', '[I3]', '[I4]', '[↩
                                   I5]', '[I6]', '[I7]')
            )
                and not exists (
                    select
                        *
                    from
                        item
                    where
                        i_custkey = c_custkey
                )
    ) as custsale
group by
    cntrycode
order by
    cntrycode;
```

Apêndice C

DDL para Criação do Ambiente *Snowflake*

É apresentado aqui o código em SQL para criação das entidades para o ambiente *snowflake*.

```
CREATE TABLE  region (
    r_regionkey INTEGER,
    r_name CHAR(25),
    r_comment VARCHAR(152)
);

CREATE TABLE  nation (
    n_nationkey INTEGER,
    n_name CHAR(25),
    n_regionkey INTEGER,
    n_comment VARCHAR(152)
);

CREATE TABLE  supplier (
    s_suppkey  INTEGER,
    s_name CHAR(25),
    s_address VARCHAR(40),
    s_nationkey INTEGER,
    s_phone CHAR(15),
    s_acctbal NUMERIC,
    s_comment VARCHAR(101)

);

CREATE TABLE  part (
    p_partkey INTEGER,
    p_name VARCHAR(55),
    p_mfgr CHAR(25),
    p_brand CHAR(10),
    p_type VARCHAR(25),
    p_size INTEGER,
    p_container CHAR(10),
```

```
    p_retailprice NUMERIC,
    p_comment VARCHAR(23)
);

CREATE TABLE  partsupp (
    ps_partkey INTEGER,
    ps_suppkey INTEGER,
    ps_availqty INTEGER,
    ps_supplycost NUMERIC,
    ps_comment VARCHAR(199)
);

CREATE TABLE  customer (
    c_custkey INTEGER,
    c_name VARCHAR(25),
    c_address VARCHAR(40),
    c_nationkey INTEGER,
    c_phone CHAR(15),
    c_acctbal NUMERIC,
    c_mktsegment CHAR(10),
    c_comment VARCHAR(117)
);

CREATE TABLE orders (
    o_orderkey BIGINT,
    o_custkey INTEGER,
    o_orderstatus CHAR(1),
    o_totalprice NUMERIC,
    o_orderdate DATE,
    o_orderpriority CHAR(15),
    o_clerk CHAR(15),
    o_shippriority INTEGER,
    o_comment VARCHAR(79)
);

CREATE TABLE  lineitem (
    l_orderkey BIGINT,
    l_partkey INTEGER,
    l_suppkey INTEGER,
    l_linenumber INTEGER,
    l_quantity NUMERIC,
    l_extendedprice NUMERIC,
    l_discount NUMERIC,
    l_tax NUMERIC,
    l_returnflag CHAR(1),
    l_linestatus CHAR(1),
    l_shipdate DATE,
    l_commitdate DATE,
```

```
    l_receiptdate DATE,
    l_shipinstruct CHAR(25),
    l_shipmode CHAR(10),
    l_comment VARCHAR(44)
);
```

Apêndice D

DDL para Criação do Ambiente *Star*

É apresentado aqui o código em SQL para criação das entidades para o ambiente *star*, adaptado do *snowflake*.

```
CREATE TABLE supplier (
    s_suppkey INTEGER NOT NULL,
    s_name CHAR(25) NOT NULL,
    s_address VARCHAR(40) NOT NULL,
    s_nationkey INTEGER NOT NULL,
    s_nation_name CHAR(25) NOT NULL,
    s_region_name CHAR(25) NOT NULL,
    s_phone CHAR(15) NOT NULL,
    s_acctbal DECIMAL(15,2) NOT NULL,
    s_comment VARCHAR(101) NOT NULL,
    s_nation_comment VARCHAR(152),
    s_region_comment VARCHAR(152)
);

CREATE TABLE customer (
    c_custkey INTEGER NOT NULL,
    c_name CHAR(25) NOT NULL,
    c_address VARCHAR(40) NOT NULL,
    c_nationkey INTEGER NOT NULL,
    c_nation_name CHAR(25) NOT NULL,
    c_region_name CHAR(25) NOT NULL,
    c_phone CHAR(15) NOT NULL,
    c_acctbal DECIMAL(15,2) NOT NULL,
    c_mktsegment  CHAR(10) NOT NULL,
    c_comment VARCHAR(117) NOT NULL,
    c_nation_comment VARCHAR(152),
    c_region_comment VARCHAR(152)
);

CREATE TABLE part (
```

```
    p_partkey     INTEGER NOT NULL,
    p_name        VARCHAR(55) NOT NULL,
    p_mfgr        CHAR(25) NOT NULL,
    p_brand       CHAR(10) NOT NULL,
    p_type        VARCHAR(25) NOT NULL,
    p_size        INTEGER NOT NULL,
    p_container   CHAR(10) NOT NULL,
    p_retailprice DECIMAL(15,2) NOT NULL,
    p_comment     VARCHAR(23) NOT NULL
);

CREATE TABLE item (
    i_itemkey INTEGER NOT NULL,
    i_partkey INTEGER NOT NULL,
    i_suppkey INTEGER NOT NULL,
    i_custkey INTEGER NOT NULL,
    i_linenumber INTEGER NOT NULL,
    i_quantity DECIMAL (15,2) NOT NULL,
    i_extendedprice DECIMAL (15,2) NOT NULL,
    i_discount DECIMAL (15,2) NOT NULL,
    i_tax DECIMAL (15,2) NOT NULL,
    i_returnflag char(1) not null,
    i_linestatus char(1) not null,
    i_shipdate date not null,
    i_commitdate date not null,
    i_receiptdate date not null,
    i_shipinstruct char(25) not null,
    i_shipmode char(10) not null,
    i_comment varchar(44) not null,
    i_orderstatus char(1) not null,
    i_ordertotalprice decimal(15,2) not null,
    i_orderdate date not null,
    i_orderpriority char(15) not null,
    i_orderclerk char(15) not null,
    i_shippriority integer not null,
    i_ordercomment varchar(79) not null,
    i_availqty integer not null,
    i_supplycost decimal(15,2)  not null,
    i_partsuppcomment varchar(199) not null
);
```

Referências Bibliográficas

[1] WREMBEL, R.; KONCILIA, C. *Data warehouses and OLAP: concepts, architectures, and solutions*. [S.l.]: Igi Global, 2007.

[2] CODD, E.; CODD, S.; SALLEY, C. Providing olap (on-line analytical processing) to user-analysis: An it mandate. *White paper*, 1993.

[3] CHAUDHURI, S.; DAYAL, U. An overview of data warehousing and olap technology. *ACM Sigmod record*, ACM, v. 26, n. 1, p. 65–74, 1997.

[4] KIMBALL, R.; ROSS, M. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. 2. ed. [S.l.]: Wiley Computer Publishing, 2002.

[5] ELMASRI, R.; NAVATHE, S. B. *Sistemas de Banco de Dados*. 6. ed. [S.l.]: Pearson Education do Brasil, 2011.

[6] Good Data Help. *Column Storage and Compression in Data Warehouse*. 2017. Acessado em: 12 de maio de 2017. Disponível em: <https://goo.gl/f1yeh8>.

[7] BOUCKAERT, S. et al. Benchmarking computers and computer networks. *EU FIRE White Paper*, 2010.

[8] TRANSACTION PROCESSING PERFORMANCE COUNCIL. *TPC Homepage*. 2017. Acessado em: 12 de agosto de 2017. Disponível em: <http://www.tpc.org>.

[9] TRANSACTION PROCESSING PERFORMANCE COUNCIL. *TPC-H Homepage*. 2017. Acessado em: 19 de maio de 2017. Disponível em: <http://www.tpc.org/tpch/>.

[10] BAX, M. P.; SOUZA, R. Modelagem estratégica de dados: Normalização versus "dimensionalização". *KMBRASIL, Anais... São Paulo: SBGC*, 2003.

[11] INMON, W. H. *Building the data warehouse*. [S.l.]: John wiley & sons, 2005.

[12] SHIM, J. P. et al. Past, present, and future of decision support technology. *Decision support systems*, Elsevier, v. 33, n. 2, p. 111–126, 2002.

[13] SEN, A.; SINHA, A. P. A comparison of data warehousing methodologies. *Communications of the ACM*, ACM, v. 48, n. 3, p. 79–84, 2005.

[14] LEVENE, M.; LOIZOU, G. Why is the snowflake schema a good data warehouse design? *Information Systems*, Elsevier, v. 28, n. 3, p. 225–240, 2003.

[15] CODD, E. F. A relational model of data for large shared data banks. *Communications of the ACM*, ACM, v. 13, n. 6, p. 377–387, 1970.

[16] POSTGRESQL: The World's Most Advanced Open Source Relational Database. 2018. Acessado em: 20 de maio de 2018. Disponível em: <https://www.postgresql.org/>.

[17] MYSQL: The world's most popular open source database. 2018. Acessado em: 20 de maio de 2018. Disponível em: <https://www.mysql.com/>.

[18] MICROSOFT SQL Server. 2018. Acessado em: 20 de maio de 2018. Disponível em: <https://www.microsoft.com/en-us/sql-server/sql-server-2017>.

[19] MICROSOFT SQL Server. 2018. Acessado em: 20 de maio de 2018. Disponível em: <https://www.sqlite.org/index.html>.

[20] MARIADB - Supporting continuity and open collaboration. 2018. Acessado em: 20 de maio de 2018. Disponível em: <https://mariadb.org/>.

[21] DATABASE and Cloud Database - Oracle. 2018. Acessado em: 20 de maio de 2018. Disponível em: <https://www.oracle.com/database/index.html>.

[22] HAN, J. et al. Survey on nosql database. In: *2011 6th International Conference on Pervasive Computing and Applications*. [S.l.: s.n.], 2011. p. 363–366.

[23] PRITCHETT, D. Base: An acid alternative. *Queue*, ACM, v. 6, n. 3, p. 48–55, 2008.

[24] SHARDING or Data Partitioning. 2018. Acessado em: 20 de maio de 2018. Disponível em: <https://www.educative.io/collection/page/5668639101419520/5649050225344512/5146118144917504>.

[25] TWITTER growth prompts switch from MySQL to 'NoSQL' database. 2010. Acessado em: 20 de maio de 2018. Disponível em: <https://www.computerworld.com/article/2520084/database-administration/twitter-growth-prompts-switch-from-mysql-to–nosql–database.html>.

[26] CHANG, F. et al. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, ACM, v. 26, n. 2, p. 4, 2008.

[27] DECANDIA, G. et al. Dynamo: amazon's highly available key-value store. In: ACM. *ACM SIGOPS operating systems review*. [S.l.], 2007. v. 41, n. 6, p. 205–220.

[28] THE Neo4j Graph Plataform - The #1 Platform for Connected Data. 2018. Acessado em: 20 de maio de 2018. Disponível em: <https://neo4j.com/>.

[29] PROJECT Voldemort: A distributed database. 2018. Acessado em: 20 de maio de 2018. Disponível em: <https://www.project-voldemort.com/voldemort/>.

[30] AMAZON DynamoDB - NoSQL Database Service. 2018. Acessado em: 20 de maio de 2018. Disponível em: <https://aws.amazon.com/dynamodb/>.

[31] RIAK. 2018. Acessado em: 20 de maio de 2018. Disponível em: <http://basho.com/products/>.

[32] REDIS. 2018. Acessado em: 20 de maio de 2018. Disponível em: <https://redis.io/>.

[33] MONGODB for Giant Ideas. 2018. Acessado em: 20 de maio de 2018. Disponível em: <https://www.mongodb.com/>.

[34] APACHE CouchDB. 2018. Acessado em: 20 de maio de 2018. Disponível em: <http://couchdb.apache.org/>.

[35] MONETDB – The column-store pioneer. 2017. Acessado em: 20 de agosto de 2017. Disponível em: <https://www.monetdb.org/Home>.

[36] C-STORE: A Column-Oriented DBMS. 2018. Acessado em: 20 de maio de 2018. Disponível em: <http://db.lcs.mit.edu/projects/cstore/>.

[37] BIGTABLE - Scalable NoSQL Database Service. 2018. Acessado em: 20 de maio de 2018. Disponível em: <https://cloud.google.com/bigtable/>.

[38] APACHE Cassandra. 2018. Acessado em: 20 de maio de 2018. Disponível em: <http://cassandra.apache.org/>.

[39] BREWER, E. A. Towards robust distributed systems. In: *PODC*. [S.l.: s.n.], 2000. v. 7.

[40] GILBERT, S.; LYNCH, N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *Acm Sigact News*, ACM, v. 33, n. 2, p. 51–59, 2002.

[41] KHOSHAFIAN, S. et al. A query processing strategy for the decomposed storage model. In: IEEE. *Data Engineering, 1987 IEEE Third International Conference on*. [S.l.], 1987. p. 636–643.

[42] MATEI, G.; BANK, R. C. Column-oriented databases, an alternative for analytical environment. *Database Systems Journal*, Academy of Economic Studies-Bucharest, Romania, v. 1, n. 2, p. 3–16, 2010.

[43] ABADI, D. J.; MADDEN, S. R.; HACHEM, N. Column-stores vs. row-stores: how different are they really? In: ACM. *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. [S.l.], 2008. p. 967–980.

[44] ABADI, D. et al. The design and implementation of modern column-oriented database systems. *Foundations and Trends® in Databases*, Now Publishers, Inc., v. 5, n. 3, p. 197–280, 2013.

[45] ABADI, D. J. et al. Materialization strategies in a column-oriented dbms. In: IEEE. *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*. [S.l.], 2007. p. 466–475.

[46] ABADI, D. J. *Query execution in column-oriented database systems*. Tese (Doutorado) — Massachusetts Institute of Technology, 2008.

[47] STONEBRAKER, M. et al. C-store: a column-oriented dbms. In: VLDB ENDOWMENT. *Proceedings of the 31st international conference on Very large data bases*. [S.l.], 2005. p. 553–564.

[48] BONCZ, P. A.; ZUKOWSKI, M.; NES, N. Monetdb/x100: Hyper-pipelining query execution. In: *Cidr*. [S.l.: s.n.], 2005. v. 5, p. 225–237.

[49] WESTMANN, T. et al. The implementation and performance of compressed databases. *ACM Sigmod Record*, ACM, v. 29, n. 3, p. 55–67, 2000.

[50] ROTH, M. A.; HORN, S. J. V. Database compression. *ACM Sigmod Record*, ACM, v. 22, n. 3, p. 31–39, 1993.

[51] ABADI, D.; MADDEN, S.; FERREIRA, M. Integrating compression and execution in column-oriented database systems. In: ACM. *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*. [S.l.], 2006. p. 671–682.

[52] BHUTTA, K. S.; HUQ, F. Benchmarking–best practices: an integrated approach. *Benchmarking: An International Journal*, MCB UP Ltd, v. 6, n. 3, p. 254–268, 1999.

[53] KYRÖ, P. Revising the concept and forms of benchmarking. *Benchmarking: An International Journal*, MCB UP Ltd, v. 10, n. 3, p. 210–225, 2003.

[54] NGAMSURIYAROJ, S.; PORNPATTANA, R. Performance evaluation of tpc-h queries on mysql cluster. In: IEEE. *Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on*. [S.l.], 2010. p. 1035–1040.

[55] NADEE, W.; NGAMSURIYAROJ, S. Performance evaluation of tpc-h queries on java ee cluster. In: IEEE. *Computer Science and Software Engineering (JCSSE), 2012 International Joint Conference on*. [S.l.], 2012. p. 385–390.

[56] THANOPOULOU, A.; CARREIRA, P.; GALHARDAS, H. Benchmarking with tpc-h on off-the-shelf hardware. *ICEIS (1)*, p. 205–208, 2012.

[57] BARATA, M.; BERNARDINO, J.; FURTADO, P. Ycsb and tpc-h: Big data and decision support benchmarks. In: IEEE. *Big Data (BigData Congress), 2014 IEEE International Congress on*. [S.l.], 2014. p. 800–801.

[58] RUTISHAUSER, N.; NOURELDIN, A. Tpc-h applied to mongodb: How a nosql database performs. Feb, 2012.

[59] SOARES, B. E. Uma avaliação experimental de desempenho entre sistemas gerenciadores de bancos de dados colunares e relacionais. 2012.

[60] TRANSACTION PROCESSING PERFORMANCE COUNCIL. *TPC BENCHMARK H Standard Specification. Revision 2.17.2*. San Francisco, 2017. Acessado em: 12 de maio de 2017. Disponível em: <https://goo.gl/uaRRcv>.

[61] RAASVELDT, M. et al. Fair benchmarking considered difficult: Common pitfalls in database performance testing. 2018.