Faculdade de Ciências e Tecnologia

Departamento de Engenharia Informática

Processamento aproximado de pesquisas para análise de Big Data

Solange de Lemos Paz

Dissertação no âmbito do Mestrado em Engenharia Informática, especialização em Sistemas de Informação orientada pelo Professor Doutor Bruno Cabral e Professor Doutor Jorge Bernardino e apresentada à Faculdade de Ciências e Tecnologia / Departamento de Engenharia Informática.

setembro de 2019

UNIVERSIDADE D
COIMBRA

Agradecimentos

Aos meus orientadores, Prof. Dr. Bruno Cabral e Prof. Dr. Jorge Bernardino pela disponibilidade e apoio sempre demonstrados na orientação de todo este trabalho.
Aos meus colegas do Departamento de Engenharia Informática pelo bom ambiente de trabalho proporcionado, companheirismo e apoio oferecido.
Ao David, por toda a força e compreensão nesta etapa importante.
A todos os meus amigos, que sempre me acompanharam.
Por fim, mas nunca menos importante, agradeço a toda a minha família, em especial aos meus pais, por todos os valores transmitidos, pela compreensão e apoio nos bons e maus momentos.

Resumo

Nos últimos dez anos o crescimento dos dados digitais aumentou exponencialmente. Com o aumento da quantidade de dados processada diariamente, a análise de dados para extrair informações relevantes de forma rápida tornou-se uma tarefa cada vez mais importante e difícil. As tecnologias atuais para análise de dados, que utilizam sistemas de bases de dados relacionais e *data warehouses* tornaram-se incapazes de lidar de forma eficiente com grandes quantidades de dados. Uma pesquisa nesses sistemas pode demorar horas até devolver um resultado, surgindo assim a necessidade de melhorar o seu desempenho, em termos de custo e tempo. Para melhorar este desempenho surgiram os sistemas de processamento aproximado de pesquisas, que garantem o processamento rápido de grandes quantidades de dados, abdicando de 100% de exatidão na resposta mas promovendo tempos de resposta mais curtos, utilizando apenas uma parte do conjunto de dados. Ao longo das últimas décadas foram propostas diversas técnicas de processamento aproximado de pesquisas, no entanto estas possuem limitações.
Neste trabalho é proposta e avaliada uma nova técnica de processamento aproximado de pesquisas que mitiga as seguintes deficiências das abordagens atuais: não requer que seja efetuada qualquer alteração na base de dados, uma vez que possui uma arquitetura de *middleware*; permite a parametrização do grau de confiança e o erro máximo admitido para a resposta de uma pesquisa e lida com a maioria dos tipos de pesquisas. Esta técnica, designada JDBCApprox, consiste na implementação de uma biblioteca Java que recorre a uma amostragem aleatória simples sem repetição para criar amostras das tabelas da base de dados e, em seguida utiliza uma base de dados com uma configuração em memória para obter uma aceleração no tempo de resposta das pesquisas. A avaliação experimental mostrou que a técnica JDBCApprox consegue ser até 24 vezes mais rápida do que o PostgreSQL e devolve na maioria dos casos respostas mais exatas do que o sistema que apresenta os melhores resultados do estado da arte.

Palavras-Chave

Redução de dados
Processamento aproximado de pesquisas
Processamento de Big Data
Amostragem

Abstract

Over the last ten years, the growth of digital data has increased exponentially. With the increase in the amount of data processed daily, using data analysis to quickly extract relevant information has become an increasingly important and difficult task. Current technologies for data analysis, which utilize relational database systems and data warehouses, have become incapable of handling large amounts of data efficiently. Performing a query on these systems may take hours before returning a result, thus emerging the need to improve their performance in terms of cost and time. To improve this performance, new processing systems of research have emerged. These systems ensure the rapid processing of large amounts of data, abdicating from 100% accuracy in the response but promoting shorter response times, using only a portion of the data set. Over the last decades, several techniques have been proposed to approximate processing of queries, however these have limitations.
This work proposes and evaluates a new technique of approximate processing of researches that mitigates the following shortcomings of current approaches: it does not require any changes to be made on the database since it has a middleware architecture; allows the parameterization of the degree of confidence and the maximum error admitted to the response of a survey and deals with most types of queries. This technique, named JDBCApprox, consists of the implementation of a Java library that uses a simple random sampling without repetition to create samples of the database tables. It then uses a database with an in-memory configuration to get an acceleration in the response time of the queries. The deployed library can be up to 24 times faster than PostgreSQL and returns, in most cases, more accurate answers than the system that presents the best state of the art results.

Keywords

Data Reduction
Approximate Query Processing
Big Data Processing
Sampling

Conteúdo

Acrónimos

API Interface de Programação da Aplicação (*Application Programming Interface*). 37, 38

AQP Processamento Aproximado de Pesquisas (*Approximate Query Processing*). xv, 1–3, 5–7, 10, 11, 13, 20, 21, 23–27, 29, 43, 67, 68

BI Inteligência do Negócio (*Business Intelligence*). 2

JDBC Conectividade de Base de Dados Java (*Java Database Connectivity)*. 37

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O presente documento descreve o trabalho desenvolvido no âmbito da unidade curricular Estágio/Dissertação do 2º ano do Mestrado em Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra (FCTUC) no ano letivo 2018/2019.
Este capítulo introdutório está dividido em quatro secções. Na primeira secção são descritos a motivação e âmbito deste trabalho e na segunda secção são explicados os seus objetivos. A terceira secção apresenta o planeamento e cronograma deste trabalho. Por fim, na última secção é explicada toda a estrutura das restantes secções.

1.1 Motivação e âmbito

Nos últimos dez anos, o crescimento dos dados digitais acelerou exponencialmente. Em cada minuto de 2018 os utilizadores do Twitter publicaram 473.400 *tweets*, os utilizadores do Instagram enviaram 49.380 fotografias e o Google realizou 3.877.140 pesquisas [3]. Com o aumento da quantidade de dados processados diariamente, a análise de dados para extrair informações valiosas de forma rápida torna-se cada vez mais importante. As tecnologias atuais para análise de dados, que utilizam sistemas de bases de dados relacionais e *data warehouses* tornaram-se incapazes de lidar de forma eficiente com grandes quantidades de dados [17]. Uma pesquisa a uma base de dados pode demorar horas, surgindo assim a necessidade de melhorar o seu desempenho, em termos de custo e tempo. Por vezes, as restrições de desempenho podem ser contornadas utilizando mais recursos computacionais. Quase todos os sistemas de análise de dados são baseados no modelo de programação de dados paralelo [23], no entanto, isso origina um custo computacional mais elevado, que nem sempre é aceitável devido aos limites de orçamento de recursos. Estes sistemas possuem dois requisitos importantes, mas contraditórios [56]: (i) alcançar tempos de resposta reduzidos e (ii) proceder a uma utilização eficiente de recursos. Para resolver a contrariedade dos requisitos dos sistemas de análises de dados, foi proposto o Processamento Aproximado de Pesquisas (*Approximate Query Processing*) (AQP), que calcula respostas aproximadas de forma muito eficiente atendendo aos requisitos de desempenho. Ao longo das últimas décadas, um grande número de técnicas de AQP [21] foram propostas, garantindo o processamento rápido de grandes quantidades de dados, abdicando de 100% de exatidão na resposta mas promovendo tempos de resposta mais curtos. O processamento é realizado recorrendo apenas a uma amostra do conjunto de dados, que garante que o resultado obtido a partir dela seja similar ao que é obtido utilizando o conjunto de dados original. Existem duas abordagens fundamentais para o processamento aproximado de pesquisas: (i) agregação online e (ii) geração offline de sinopses, ou seja, uma amostra dos dados. Nos sistemas de agregação online os utilizadores conseguem observar o progresso das respostas das pesquisas e controlar a execução das mesmas em tempo real. Além disso,

também podem estabelecer requisitos de latência e limites de erro. Com isso estabelecido, a pesquisa é executada até o limite de erro ser atingido ou o tempo de latência ser esgotado. Nos sistemas de geração offline de sinopses, são utilizados os dados das pesquisas executadas anteriormente para responder a pesquisas futuras de forma eficiente. Apesar destas duas abordagens serem distintas, possuem problemas semelhantes relacionados com o seu modo de funcionamento [36]. A maioria dos sistemas de AQP exigem que seja realizada alguma modificação nos mecanismos de bases de dados existentes. Outro problema está relacionado com a incompatibilidade com ferramentas de Inteligência do Negócio (*Business Intelligence*) (BI), uma vez que as estimativas de erro devolvidas pelos sistemas de AQP não são compatíveis com elas. O último problema diz respeito ao *design* de interfaces dos utilizadores, pois estes consideram difícil interpretar o erro da resposta aproximada. Para mitigar estes problemas surgiu o JDBCApprox, que consiste na implementação de uma biblioteca Java que permita realizar AQP com base numa amostragem aleatória sem repetição. As amostras são criadas na base de dados de origem e em seguida são copiadas para uma base de dados com configuração em memória, não sendo necessário realizar qualquer alteração nos mecanismos de base de dados. Além disso, esta biblioteca pode ser facilmente utilizada com uma interface para o utilizador, permitindo que seja simples a sua utilização e a compreensão dos resultados obtidos.

1.2 Objetivos e abordagens

Os principais objetivos desta dissertação são:

1. Estudar as técnicas existentes para o processamento aproximado de pesquisas e identificar as suas limitações.
2. Desenvolver uma biblioteca que permita realizar pesquisas aproximadas em Java.
3. Avaliar a biblioteca desenvolvida e compará-la com as técnicas existentes para o processamento aproximado de pesquisas.

1.3 Planeamento e cronograma

Como ferramenta de gestão de projeto, foram criados diagramas de *Gantt* para o primeiro e segundo semestre. O presente trabalho teve início a 24 de setembro de 2018 e terminará a 11 de setembro de 2019.

1.3.1 Primeiro semestre

Durante o primeiro semestre o trabalho baseou-se na análise do estado da arte. Todas as semanas ocorreram reuniões com os orientadores, que consistiam na apresentação do trabalho desenvolvido durante a semana, no esclarecimento de questões que surgiam e no planeamento para a semana seguinte. Foi publicado um artigo científico na conferência "17th LACCEI International Multi-Conference for Engineering, Education, and Technology", com o título of Approximate Query Processing Systems" que avalia duas ferramentas estudadas durante o desenvolvimento do estado da arte: VerdictDB e XDB. Este artigo encontra-se no Anexo A.
A Figura 1.1 apresenta o diagrama de *Gantt* referente ao primeiro semestre.

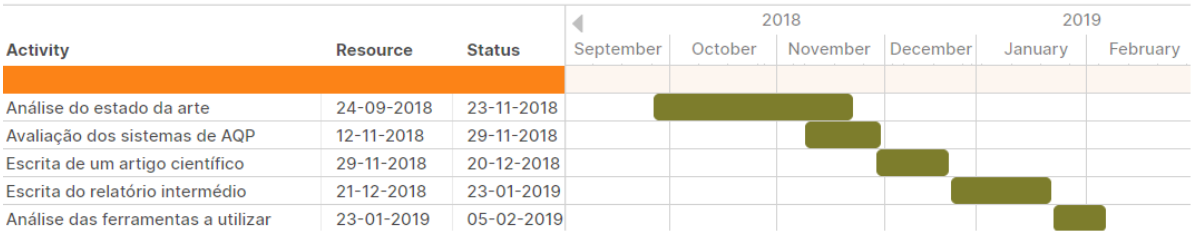

Figura 1.1: Diagrama de *Gantt* referente ao primeiro semestre

1.3.2 Segundo semestre

No primeiro semestre tinha sido planeado a implementação de uma biblioteca que suportasse o processamento aproximado de pesquisas sobre um ORM Java, o Speedment [9]. Essa implementação foi realizada, no entanto não obteve os resultados que pretendíamos. Ou seja, executar uma pesquisa com essa biblioteca era mais rápido do que se a mesma pesquisa fosse executada no Speedment, no entanto não era mais rápido comparado com outras abordagens existentes. Este problema estava relacionado com o facto do Speedment apenas melhorar o seu desempenho na execução de pesquisas quando é utilizada a versão paga, sendo na versão *opensource* mais lento executar uma pesquisa com o Speedment do que diretamente num motor de base de dados.
Foi necessário criar uma nova estratégia, sendo que o trabalho do segundo semestre baseou-se no desenvolvimento de uma biblioteca Java, que permite obter respostas aproximadas de uma pesquisa com base numa amostragem aleatória. Esse desenvolvimento passou por diversas etapas: (i) especificação da biblioteca, que inclui o desenho da arquitetura, (ii) implementação da biblioteca, que consiste na integração de uma amostragem aleatória simples sem repetição para criar sinopses dos dados, juntamente com um processamento em memória, (iii) validação, que se foca num estudo comparativo com o sistema de AQP que apresentava os melhores resultados do estado da arte.
A Figura 1.2 apresenta o diagrama de *Gantt* referente ao segundo semestre.

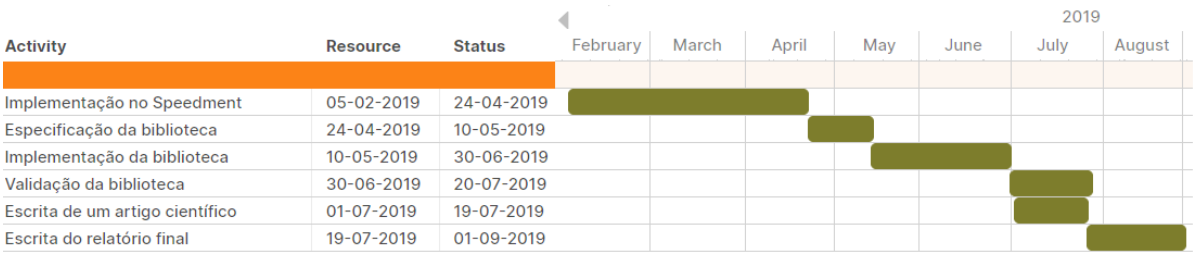

Figura 1.2: Diagrama de *Gantt* referente ao segundo semestre

1.4 Estrutura do documento

A parte restante desta dissertação encontra-se dividida em cinco capítulos. O segundo capítulo apresenta os conceitos que estão na base do processamento aproximado de pesquisas. O terceiro capítulo descreve os sistemas de processamento aproximado de pesquisas mais recentes, onde no final é realizada uma comparação entre esses sistemas e são descritos os problemas que estes apresentam. O quarto capítulo a abordagem proposta para o processamento aproximado de pesquisas. O capítulo cinco apresenta a avaliação experimental realizada, onde no final são discutidos os resultados. Por último, o capítulo seis apresenta as conclusões da presente dissertação.

Capítulo 2

Conceitos Base

Este capítulo descreve os conceitos que estão na base do processamento aproximado de pesquisas. Inicialmente é apresentada a definição do processamento de pesquisas aproximado, que possui duas abordagens: (i) agregação online e (ii) geração offline de sinopses. Em seguida são apresentados os quatro métodos que podem ser utilizados em sistemas de AQP para gerar sinopses de dados: (i) amostras, (ii) histogramas, (iii) *wavelets* e (iv) *sketches*.

2.1 Processamento aproximado de pesquisas

No contexto de bases de dados, o processamento de pesquisas consiste no processo que recolhe as informações disponíveis nas bases de dados [38]. Como a quantidade de dados que estas possuem é muito grande, torna-se difícil processar pesquisas de forma rápida e eficiente. Na maior parte dos casos, é impossível ou muito dispendioso obter respostas exatas num curto intervalo de tempo. Para contornar estes problemas, surgiu o AQP, que devolve respostas aproximadas, com garantia de qualidade e de forma eficiente.
A Figura 2.1 ilustra a arquitetura geral para os sistemas de AQP [38]. Existem dois componentes principais nessa arquitetura: (i) um componente que utiliza as tabelas da base de dados para construir sinopses e (ii) um componente que reescreve uma pesquisa para posteriormente ser respondida aproximadamente através das sinopses, devolvendo uma resposta com uma estimativa de erro.

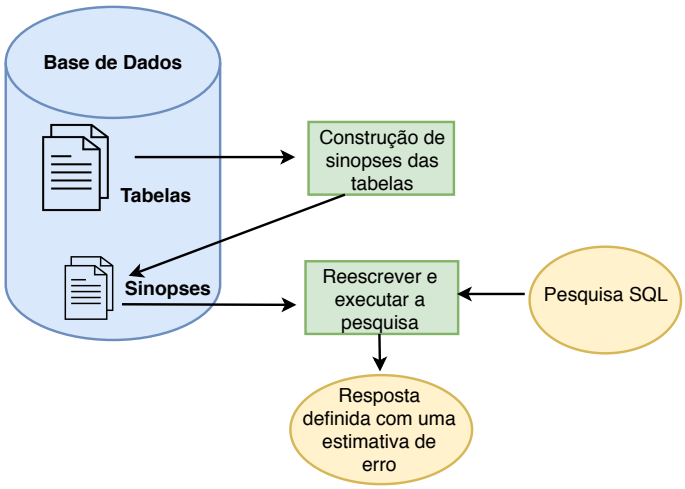

Figura 2.1: Processamento aproximado de pesquisas - Adaptado de [22]

O processamento aproximado de pesquisas é projetado essencialmente para pesquisas agregadas, como é o caso de pesquisas que utilizam as funções COUNT, SUM e AVERAGE. As respostas a essas pesquisas são acompanhadas por um erro relativo. O erro relativo de uma determinada pesquisa Q pode ser quantificado através da seguinte fórmula:

$$\textit{Erro relativo da pesquisa (Q)} = \left| \frac{\textit{Resposta Exata - Resposta Aproximada}}{\textit{Resposta Exata}} \right| \tag{2.1}$$

Os sistemas de AQP recorrem a diversos métodos para reduzir a quantidade de dados processados, uma vez que é difícil utilizar a totalidade de um conjunto de dados quando o seu tamanho é muito grande. Este problema pode ser contornado através da criação de uma sinopse que é utilizada para analisar os dados. A ideia base das sinopses de dados é reduzir o conjunto de dados completo [30]. Uma sinopse é um substituto dos dados, que é consultada em vez do conjunto de dados completo, recolhendo as propriedades principais dos dados originais e ocupando menos espaço. Quando uma sinopse é utilizada num mecanismo de AQP, a pesquisa do utilizador é executada com base nos dados da sinopse, em vez dos dados originais. Os quatro principais tipos de sinopses são [21]: (i) amostras, onde inicialmente é selecionado um pequeno número de elementos do conjunto de dados e, em seguida algumas estatísticas como a média e variância da amostra são calculadas, sendo utilizadas para estimar o resultado da pesquisa; (ii) histogramas, onde para cada intervalo do conjunto de dados são calculadas as estatísticas mais representativas, que posteriormente são utilizadas para restabelecer o valor de todo o conjunto de dados nesse intervalo; (iii) *wavelets*, que consistem na divisão dos dados em diferentes componentes de frequência, para serem analisados numa janela de menor resolução e (iv) *sketches*, onde grandes quantidades de dados são processados e o *sketch* é atualizado de forma contínua e rápida. As amostras são as mais utilizadas nos atuais sistemas de AQP [33].
O processamento aproximado de pesquisas possui duas abordagens fundamentais [36]: (i) agregação online e (ii) geração offline de sinopses.

2.1.1 Agregação online

Num sistema que suporta o processamento de agregação online, os utilizadores podem observar o progresso das pesquisas de agregação e controlar a execução das mesmas em tempo real [31]. O processamento de agregação online é iterativo, uma vez que está continuamente a melhorar a precisão das respostas através da consulta de mais amostras, até que tenha sido processado o conjunto de dados completo ou seja interrompido o seu processamento [55]. Num determinado momento do cálculo, se o utilizador estiver satisfeito com os resultados atuais, poderá cancelar o processamento, de modo a minimizar o seu custo. Se o utilizador não especificar a sua preferência, a pesquisa será executada até ao final e serão devolvidos os resultados utilizando todos os dados.

2.1.2 Geração offline de sinopses

Se o *workload* de uma determinada pesquisa não se alterar dinamicamente, é possível criar sinopses com base no *workload* da pesquisa executada anteriormente, utilizando-as para responder a pesquisas futuras de forma eficiente [36]. Essas sinopses são compostas por duas fases: fase de construção e fase de resposta da pesquisa. Na fase inicial, a sinopse de dados é criada e armazenada para a fase de resposta da pesquisa e na segunda fase, é retornada uma resposta aproximada utilizando a sinopse de dados.

2.2 Sinopse de dados

Uma sinopse de um conjunto de dados captura as propriedades dos dados originais, gerando conjuntos de dados menores [21]. A utilização de sinopses é essencial para gerir grandes quantidades de dados, pois na maioria das vezes é impossível lidar com a sua totalidade. Em vez disso, é muito mais conveniente criar uma sinopse e, em seguida, utilizar essa sinopse para analisar os dados. Em seguida são apresentados os tipos de sinopses de dados mais utilizadas para grandes quantidades de dados.

2.2.1 Amostras

Para responder a questões de pesquisa, é difícil recolher a totalidade dos itens do conjunto de dados, uma vez que esta tarefa consome muito tempo e muitos recursos [50]. Assim, existe a necessidade de selecionar uma amostra, sendo aplicadas técnicas de amostragem para reduzir a quantidade de dados.
A amostragem de dados consiste em selecionar parte dos elementos do conjunto de dados, de modo a construir um subconjunto denominado amostra, sendo que o tamanho desta deverá ser menor do que o número total de elementos do conjunto original [20]. A amostra deve ser representativa, o que significa que deve refletir as caraterísticas importantes [32], de modo a que os resultados obtidos sejam praticamente os mesmos do que quando se utiliza a totalidade do conjunto de dados. Essas amostras podem ser usadas para fornecer rapidamente respostas aproximadas em pesquisas.
Nas últimas décadas aumentaram os trabalhos de investigação relacionados com o AQP baseado em amostragem [14], [15]. Para tabelas pequenas, desenhar uma amostra pode ser um processo rápido e direto. No entanto, para tabelas maiores, que podem ter um tamanho superior ao disponível na memória, ou a sua totalidade pode não estar armazenada em disco, são necessárias técnicas mais avançadas para tornar o processo de amostragem escalável. Para os sistemas de AQP existem quatro tipos de amostragem principais [47]: (i) amostragem de Bernoulli, (ii) amostragem aleatória simples, (iii) amostragem estratificada e (iv) amostragem de reservatório.

Amostragem de Bernoulli

A amostragem de Bernoulli é um método de amostragem no qual todos os itens do conjunto de dados original têm a mesma probabilidade de serem selecionados e as variáveis de inclusão são conjuntamente independentes [19]. Cada elemento da população é submetido a um teste independente de Bernoulli que determina se o elemento se torna parte da amostra. Para cada item do conjunto de dados é gerado um número aleatório no intervalo [0, 1] e, se o número aleatório for menor do que a fração de amostragem, o item do conjunto de dados é adicionado à amostra, caso contrário, é ignorado. Um exemplo da aplicação da amostragem de Bernoulli é no lançamento de uma moeda, para observar se ocorre cara ou coroa. Os dois eventos possíveis: sair cara e sair coroa, são denominado como sucesso e insucesso. O teste de Bernoulli é caraterizado por uma variável aleatória X, definida como X=1 em caso de sucesso e X=0 em caso de insucesso.

Amostragem aleatória simples

A amostragem aleatória simples permite obter k itens de dados distintos, onde k é o tamanho da amostra, do conjunto de dados originais, de modo a que todas as combinações possíveis de k itens sejam igualmente prováveis de serem a amostra selecionada [32]. Na amostragem aleatória simples, todos os itens do conjunto de dados têm igual probabilidade de serem incluídos na amostra selecionada.

Existem dois tipos de amostragem aleatória simples: (i) amostragem aleatória simples com repetição e (ii) amostragem aleatória simples sem repetição. A amostragem sem repetição consiste em selecionar uma amostra de um conjunto de dados, de modo a que todos os elementos do conjunto tenham a mesma probabilidade de serem selecionados. Um elemento que foi selecionado não regressa ao conjunto de dados para ser disponibilizado novamente para a amostragem. Por sua vez, a amostragem com repetição consiste em selecionar uma amostra de um conjunto de dados, de modo a que todos os elementos do conjunto tenham a mesma probabilidade de serem selecionados. Um elemento que foi selecionado regressa ao conjunto de dados para ser disponibilizado novamente para a amostragem.

Amostragem estratificada

A amostragem estratificada traduz-se na divisão do conjunto de dados em subconjuntos, designados estratos. Através de cada estrato é desenhada uma amostra aleatória, que posteriormente é combinada para construir a amostra dos dados originais [51]. A amostragem estratificada assegura que os itens de dados de cada estrato devem estar presentes na amostra e nenhum é ignorado.
Existem dois tipos de amostragem estratificada [47]: (i) amostragem estratificada proporcional e (ii) amostragem estratificada desproporcional. No primeiro tipo, a mesma fração de amostragem é aplicada para cada estrato, ou seja, o tamanho da amostra de cada estrato é proporcional ao tamanho do estrato. Por sua vez, na amostragem estratificada desproporcional são aplicadas diferentes frações de amostragem para diferentes estratos.

Amostragem de reservatório

A amostragem de reservatório [34] é utilizada para escolher aleatoriamente amostras de um conjunto de dados, onde o total de itens é muito grande ou desconhecido. Normalmente esse total de itens é grande o suficiente para que a lista não caiba na memória principal. Isto origina uma grande matriz/fluxo de dados para simplificar, sendo necessário selecionar aleatoriamente itens desse conjunto de dados. Um solução para esta abordagem é criar um reservatório/array, que será preenchido com esses itens de dados. Cada item do fluxo é selecionado aleatoriamente e colocado no reservatório de acordo com o seguinte: se o item selecionado não tiver sido selecionado anteriormente, deve ser colocado no reservatório.
A amostragem de reservatório recebe então itens de um conjunto de dados de um fluxo e mantém uma amostra num *buffer* chamado reservatório. Com a amostragem de reservatório não é necessário conhecer qual o número total de itens no fluxo, garantindo também que cada item no fluxo tenha a mesma probabilidade de ser selecionado para o reservatório.

Vantagens e desvantagens dos tipos de amostragem

A Tabela 2.1 apresenta as principais vantagens e desvantagens dos tipos de amostragem abordados na secção anterior.

Tabela 2.1: Vantagens e desvantagens dos tipos de amostragem

	Vantagens	**Desvantagens**
Amostragem de Bernoulli	- Todos os itens do conjunto de dados têm igual probabilidade de serem selecionados. - Não existe a probabilidade de um item do conjunto de dados voltar a ser selecionado.	- Pode produzir amostras demasiado grandes. - Requer que o tamanho do conjunto de dados seja conhecido antecipadamente.
Amostragem Aleatória Simples	- Permite que sejam realizadas inferências estatísticas sobre o conjunto de dados. - Probabilidade elevada dos itens representarem todo o conjunto de dados.	- É fácil obter os dados errados, assim como é fácil acertar. - Requer que o tamanho do conjunto de dados seja grande.
Amostragem Estratificada	- Permite realizar generalizações da amostra para a população. - Fornece uma amostra que é altamente representativa da população.	- Não é útil quando a população não pode ser dividida em subgrupos separados. - Consome muito tempo, especialmente ao criar amostras maiores.
Amostragem de Reservatório	- Não é necessário conhecer o tamanho do conjunto de dados. - É possível amostrar todos os itens representativos do conjunto de dados.	- Pode produzir amostras demasiado pequenas. - Não funciona bem quando é necessário selecionar um grande subconjunto de dados.

Na amostragem de Bernoulli todos os itens do conjunto de dados têm uma probabilidade igual de serem selecionados, o que origina uma amostra mais completa, uma vez que pode possuir grande parte dos itens do conjunto de dados. Como cada item é analisado de forma individual, podem ocorrer situações onde as amostras produzidas são demasiado grandes e não se justifica a sua utilização. Neste tipo de amostragem não existe a probabilidade de um item do conjunto de dados voltar a ser selecionado, uma vez que não ocorre repetição dos itens. Isto origina uma amostra mais realista quando comparada com o conjunto de dados. É ainda necessário que seja conhecido o tamanho do conjunto de dados antes de ser realizada a amostragem, no entanto esse tamanho pode não ser conhecido e amostragem não é aplicada.
A amostragem aleatória simples utiliza números aleatórios, o que garante que as amostras

variem de acordo com o conjunto de dados. Assim, a probabilidade dos itens representarem todo o conjunto de dados é elevada. Como todo o processo desta amostragem está associado à aleatoriedade, os itens de dados selecionados para a amostra podem ser certos, quando representam bem o conjunto de dados, ou errados, quando são muito diferentes do conjunto de dados. No entanto, esta aleatoriedade permite que sejam realizadas generalizações dos itens para o conjunto de dados. A amostragem aleatória simples requer que o conjunto de dados seja grande, para permitir que sejam selecionados mais itens aleatoriamente.
A amostragem estratificada assegura que todos os itens do conjunto de dados são possíveis para a amostra, garantindo que nenhum é ignorado e fornecendo uma amostra que é altamente representativa do conjunto de dados. Como nenhum item é ignorado, a amostragem estratificada pode consumir muito tempo, especialmente na criação de amostras maiores. Esta amostragem necessita de dividir o conjunto de dados em subconjuntos menores, não sendo possível a sua aplicação a conjuntos de dados impossíveis de dividir. Como todos os itens de dados são agrupados, é possível realizar generalizações da amostra para o conjunto de dados.
A amostragem de reservatório pode ser utilizada quando o tamanho do conjunto de dados é desconhecido. Como cada item que não for selecionado é novamente colocado para amostrar enquanto o reservatório tiver espaço livre, é possível que seja amostrada uma grande quantidade de itens representativos. A amostragem de reservatório pode produzir amostras demasiado pequenas, pois os itens são eliminados do reservatório. Esta amostragem assume que a amostra é compatível com a memória principal do reservatório, no entanto, quando é necessário selecionar um grande subconjunto de dados é necessário utilizar outros métodos de amostragem.

2.2.2 Histogramas

Um histograma tem como função resumir o conjunto de dados, dividindo-o em vários intervalos baseados em valores de uma coluna numérica [46]. Para cada intervalo, o histograma calcula as estatísticas mais representativas que podem ser utilizadas para reconstruir o valor de todo o conjunto de dados nesse intervalo, por exemplo, armazena os limites inferior e superior desse intervalo e em seguida conta os números presentes nesse intervalo.
Os histogramas permitem responder de forma rápida às pesquisas que utilizam funções agregadas durante um determinado intervalo [21]. Podendo também ser utilizados em pesquisas com junções. Vários resultados teóricos indicam que não se deve esperar que os histogramas forneçam respostas precisas de acordo com uma pesquisa. No entanto, devido à sua simplicidade eles continuam a ser utilizados para uma ampla variedade de pesquisas.

2.2.3 Wavelets

As *wavelets* [28] são transformações localizadas e ortogonais que permitem uma visão de resolução múltipla dos dados em mais do que uma dimensão. Para essa visão, elas dividem os dados em diferentes componentes de frequência e, em seguida, estudam cada componente numa janela de menor resolução [27].
A transformada *wavelet* de um conjunto de dados consiste numa aproximação geral juntamente com os coeficientes que influenciam o conjunto de dados. No AQP as *wavelets* podem ser utilizadas para estimar a resposta de acordo com o domínio dos coeficientes das *wavelets*, por exemplo em junções [21]. Como as *wavelets* são geralmente transformações lineares simples, permitem que sejam construídos algoritmos eficientes para a construção de sinopses.

2.2.4 Sketches

Os *sketches* [21] são utilizados para a transmissão de dados em tempo real, onde grandes quantidades de dados são transmitidas e o *sketch* é continuamente atualizado de forma rápida. Os *sketches* são projetados com o objetivo da atualização causada por um novo dado ser independente dos dados que já foram processados.
Os algoritmos de *streaming* normalmente criam uma sinopse compacta dos dados observados, que é menor do que a totalidade dos dados. Cada atualização na *stream* altera essa sinopse, fazendo com que esta a qualquer momento possa fornecer uma resposta aproximada a uma determinada pesquisa. Os *sketches* fornecem amostras aos sistemas de AQP, permitindo que estes possam produzir respostas aproximadas em tempo real.

2.3 Resumo

Neste capítulo foram apresentados os conceitos que estão na base do AQP, nomeadamente as duas abordagens que o processamento aproximado de pesquisas possui e os métodos que podem ser utilizados nos sistemas de AQP para gerar sinopses de dados.
A agregação online é utilizada quando estão a ser analisadas grandes quantidades de dados que demoram muitas horas, permitindo que o utilizador consiga visualizar um resultado aproximado antes do final da execução da pesquisa. Em contrapartida, a geração offline de sinopses utiliza-se quando o *workload* de uma determinada pesquisa não se altera dinamicamente, uma vez que permite a criação de sinopses através do *workload* da pesquisa executada anteriormente. Estas sinopses são utilizadas para responder a futuras pesquisas, de forma eficiente.
Atualmente, para a geração de sinopses de dados são utilizadas maioritariamente técnicas de amostragem, uma vez que estas podem ser utilizadas para dados correlacionados e não uniformes. Além disso, os métodos de estimativa de erro também podem ser utilizados de forma eficaz com técnicas de amostragem. Embora estas técnicas sejam flexíveis e rápidas, por vezes não conseguem lidar com certos tipos de pesquisas, como as que utilizam as funções COUNT e DISTINCT.

Capítulo 3

Estado da Arte

Este capítulo tem como objetivo apresentar o estado da arte e os trabalhos relacionados com os sistemas de processamento aproximado de pesquisas existentes.
Atualmente as redes sociais e os dispositivos móveis continuam a criar grandes volumes de dados. Dada a abundância dos conjuntos de dados com informações relevantes, a extração de conhecimento tornou-se uma tarefa com elevada importância [40]. As ferramentas de análise de dados atuais, como os mecanismos de bases de dados tradicionais, são um obstáculo para muitas atividades [41]. Estes mecanismos podem demorar horas para devolver uma resposta a uma pesquisa, quando são confrontados com conjuntos de dados suficientemente grandes [41]. Quando executam um grande número de pesquisas em simultâneo num *cluster* partilhado, os mecanismos de bases de dados são altamente utilizados, diminuindo a sua eficiência.
Nos últimos anos foram publicados diversos trabalhos de investigação que apresentam propostas de sistemas que suportam o AQP. Estes sistemas são apresentados a seguir.

3.1 XDB

As junções consomem muitos recursos e a agregação online é uma abordagem eficaz para explorar a relação entre eficiência e precisão de pesquisas de forma contínua e online. Para suportar esta ideia, surgiu o XDB [35] que integra o *Wander Join* na versão mais recente do PostgreSQL. O *Wander Join* consiste em selecionar aleatoriamente uma linha de uma tabela e, em seguida, realiza uma "caminhada aleatória"a partir da linha recolhida. Para cada etapa dessa caminhada apenas são consideradas as linhas da outra tabela que se podem unir a ela. Por exemplo, assumindo três tabelas T1, T2 e T3 com os seguintes atributos [T1(A,B), T2(B,C) e T3(C,D)]. Supondo que T2 tem um índice no atributo B, T3 possui um índice no atributo C e a função de agregação é SUM(D), um caminho pode ser amostrado aleatoriamente escolhendo ao acaso um vértice em T1 e andando aleatoriamente em direção a T3. Em cada passo da "caminhada aleatória", se o vértice atual tiver vizinhos na tabela a seguir, é escolhido um ao acaso para realizar a caminhada até ele, sendo extraídas amostras das linhas que se podem unir.
Para pesquisas com várias junções, podem existir diferentes caminhos. Para contornar este problema, os autores implementaram um otimizador de planos de caminhada, que enumera todos os possíveis planos de caminhada e em seguida realiza aproximadamente 100 passeios experimentais aleatórios para cada plano. Depois de realizados os passeios, é medida a variância de cada plano e é selecionado o melhor.
A Figura 3.1 apresenta a arquitetura interna do PostgreSQL, que é semelhante à arquitetura do XDB. O XDB altera o *SQL Parser*, o *Query Optimizer*, que inclui o otimizador de planos de caminhada, e o *Query Executor* para suportar palavras utilizadas na pesquisa

como ONLINE, CONFIDENCE, WITHINTIME e REPORTINTERVAL. Estas palavras são utilizadas respetivamente para referir que se trata de uma pesquisa online, para definir o intervalo de confiança, para estabelecer o requisito máximo de latência e para indicar qual o intervalo de tempo em que são devolvidos resultados.

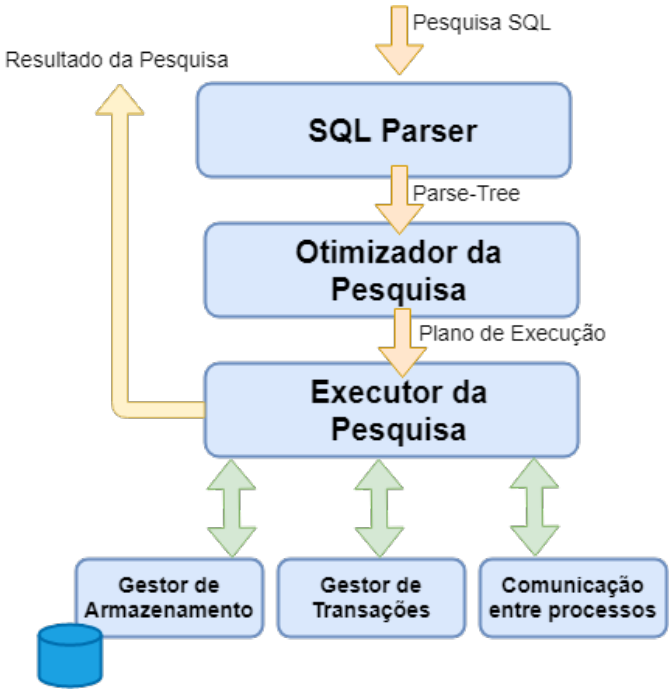

Figura 3.1: Arquitetura interna do PostgreSQL/XDB - Adaptado de [5]

As pesquisas são enviadas para um *Query Optimizer* que cria um *join query graph* e gera percursos de caminhada a partir do mesmo. Este também substitui os operadores de agregação por estimadores de agregação *online* e operadores de intervalo de confiança. Se a pesquisa possuir a cláusula INITSAMPLE, são executadas várias avaliações utilizando os vários caminhos para selecionar o melhor plano de caminhada. O *Query Executor* percorre todos os planos de caminhada, executa várias execuções de teste e calcula uma estimativa de taxa de rejeição e uma variância. Em seguida, os planos de caminhada são ordenados pela taxa de rejeição e é utilizada a variância em caso de empate. O *Query Executor* extrai amostras através dos índices *B-Tree* para cada caminho. Em seguida, os predicados de seleção são aplicados imediatamente quando as linhas relacionadas são amostradas. Quando a caminhada está completa, o *Query Executor* retorna os estimadores atuais e os intervalos de confiança periodicamente. No final é devolvida uma linha vazia quando o requisito de latência é esgotado, que informa ao PostgreSQL que não há mais linhas disponíveis.
A Tabela 3.1 apresenta as principais vantagens e desvantagens do XDB. O algoritmo do *Wander Join*, que é utilizado pelo XDB, pode ser facilmente manipulado iniciando as caminhadas aleatórias a partir dessa tabela, o que melhora o seu desempenho. Além disso, também possui um otimizador que seleciona o melhor plano de caminhada, sem ser necessário recolher qualquer dado estatístico antes. O XDB armazena as relações entre tabelas e os índices na memória principal, que irá afetar o seu desempenho caso existam muitas tabelas de dados. É difícil utilizar o XDB em várias máquinas, uma vez que a paralelização do plano de execução não é simples, pois foi projetada para funcionar em *single-threaded*.

Tabela 3.1: Vantagens e desvantagens do XDB

Vantagens	Desvantagens
- Se os atributos GROUP BY forem de uma única tabela, o *wander join* pode ser facilmente manipulado iniciando as caminhadas aleatórias a partir dessa tabela. - Possui um otimizador que escolhe o melhor plano de caminhada, sem ter que recolher nenhum dado estatístico antes.	- Armazena relações e índices na memória principal. - O *wander join* é *single-threaded* e a paralelização do procedimento de otimização do plano de execução não é simples.

3.2 ApproxJoin

O *ApproxJoin* [48] foi desenvolvido para reduzir a sobrecarga de operações de junções distribuídas em sistemas de análise de big data, como o Apache Flink e Apache Spark, funcionando em duas fases. A primeira fase consiste na filtragem de itens redundantes, sendo utilizado um filtro Bloom [18], que consiste numa estrutura de dados probabilística utilizada para testar se um elemento pertence a um conjunto de dados. O filtro Bloom é utilizado para eliminar os dados que não participam na junção, reduzindo a transferência de dados a executar na junção. Essa filtragem de dados acontece em paralelo em cada nó que armazena partições de entrada e simultaneamente em todas as tabelas de entrada. A segunda fase consiste na produção de uma resposta aproximada em junções distribuídas, onde é utilizada uma amostragem estratificada para esse efeito. Essa amostragem é utilizada dentro do processo da junção: enquanto o produto vetorial está a ser calculado, os conjuntos de dados são amostrados. A combinação das duas técnicas permite que os utilizadores obtenham amostras aleatórias imparciais sobre as junções.
O *ApproxJoin* executa os seguintes passos para obter uma aproximação em junções distribuídas, sendo que taxa de amostragem é selecionada com base na precisão e nos requisitos de latência definidos pelo utilizador:

1. **Determinar os parâmetros de amostragem**
   É utilizada uma função de custo para calcular o tamanho ideal da amostra de acordo com o orçamento fornecido pelo utilizador (requisitos de latência e intervalo de confiança). Este cálculo garante que a pesquisa é executada sem ultrapassar esse orçamento.
2. **Amostras e execução da pesquisa**
   Utilizando um parâmetro de taxa de amostragem, o *ApproxJoin* cria amostras durante a junção e executa a pesquisa de agregação utilizando as amostras obtidas.
3. **Estimativa de erro**
   Após a pesquisa ser executada, é fornecido um resultado aproximado, juntamente com a estimativa de erro correspondente.

A Figura 3.2 ilustra o modo de funcionamento do ApproxJoin. Este utiliza um filtro Bloom para evitar o que os itens de dados se misturem e, em seguida, aplica a amostragem estratificada para obter uma amostra representativa do resultado da junção.

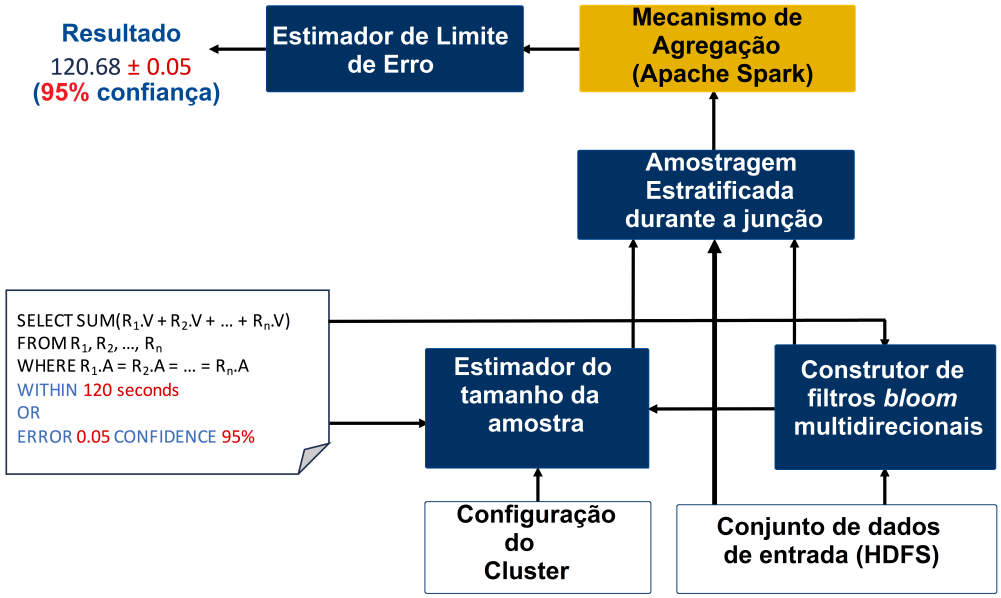

Figura 3.2: Modo de funcionamento do ApproxJoin - Adaptado de [1]

A Tabela 3.2 apresenta as principais vantagens e desvantagens do ApproxJoin. O ApproxJoin recorre a filtros Bloom para reduzir a quantidade de dados a amostra. Estes filtros testam se um elemento pertence a um conjunto de dados, onde para cada elemento se utilizam diversas funções de *hash*. Cada uma dessas funções indicará um elemento que deve pertencer à amostra. Esta amostra possui todas as chaves de junção, mesmo aquelas que se referem a poucos itens de dados das tabelas, garantindo que todas as categorias dos dados são consideradas. O facto do ApproxJoin funcionar apenas em sistemas de bases de dados distribuídos torna-o limitado.

Tabela 3.2: Vantagens e desvantagens do ApproxJoin

Vantagens	Desvantagens
- Reduz a quantidade de dados antes de realizar a amostragem. - A amostra final possui todas as chaves de junção (*join keys*), mesmo aquelas que se referem a poucos itens de dados.	- É limitado, funcionando apenas em sistemas de bases de dados distribuídos.

3.3 BlinkDB

O *BlinkDB* [16] é um mecanismo com funcionamento paralelo desenvolvido para executar pesquisas em grandes volumes de dados, devolvendo uma resposta aproximada. Este mecanismo estende a estrutura do Apache Hive [52], adicionando a este dois componentes principais: um módulo de amostragem e um módulo de seleção de amostras. O módulo de amostragem tem como função criar e manter amostras dos dados ao longo do tempo. Por sua vez, o módulo de seleção de amostras tem como objetivo calcular um perfil de latência de erro para as pesquisas.
No *BlinkDB* é criado um conjunto de amostras para permitir respostas precisas e rápidas

a cada pesquisa, utilizando uma amostragem estratificada num determinado conjunto de colunas. Apesar desta amostragem ser útil, os seus custos de armazenamento são elevados, sendo possível construir apenas um número limitado de amostras estratificadas. Assim, no *BlinkDB* foi resolvido um problema de otimização de modo a decidir quais os conjuntos de colunas nos quais são construídos amostras. Para essa decisão, o problema de otimização tem em consideração três fatores: a dispersão dos dados; as características do *workload* e o custo de armazenamento das amostras.

A escolha de uma amostra apropriada para uma determinada pesquisa depende do conjunto de colunas que ocorrem nas cláusulas WHERE e GROUP BY. Se forem encontradas uma ou mais amostras estratificadas no mesmo conjunto de colunas, é selecionada para executar a que possuir um menor número de colunas. No entanto, se não houver nenhuma amostra estratificada num conjunto de colunas, a pesquisa é executada em paralelo nos subconjuntos de memória de todas as amostras atualmente presentes no sistema. Dessas amostras é selecionada a que possui uma maior seletividade em comparação com outras, onde a seletividade é definida como a razão entre o número de linhas selecionadas pela pesquisa e o número de linhas na amostra.

Quando é decidido qual o conjunto de amostras a utilizar, é necessário selecionar uma amostra em particular e escolher uma sub-amostra de tamanho apropriado, com base nas restrições de latência e erro definidas pelo utilizador. Para selecionar a amostra em particular, o *BlinkDB* possui um *error-latency profile*, que define a taxa na qual o erro diminui, com o aumento dos tamanhos da amostra. Este perfil é construído através da execução da pesquisa em amostras menores, sendo possível estimar assim a latência e o erro para amostras maiores.

A Figura 3.3 ilustra o modo de funcionamento do BlinkDB. Este possui um módulo de amostragem que cria amostras uniformes e estratificadas a partir de uma tabela de dados. Quando uma pesquisa entra no sistema, o módulo de seleção de amostras seleciona uma amostra de tamanho apropriado com base nos requisitos de latência e limites de erro. No final, as pesquisas são executadas em paralelo nas amostras escolhidas e as respostas são devolvidas ao utilizador.

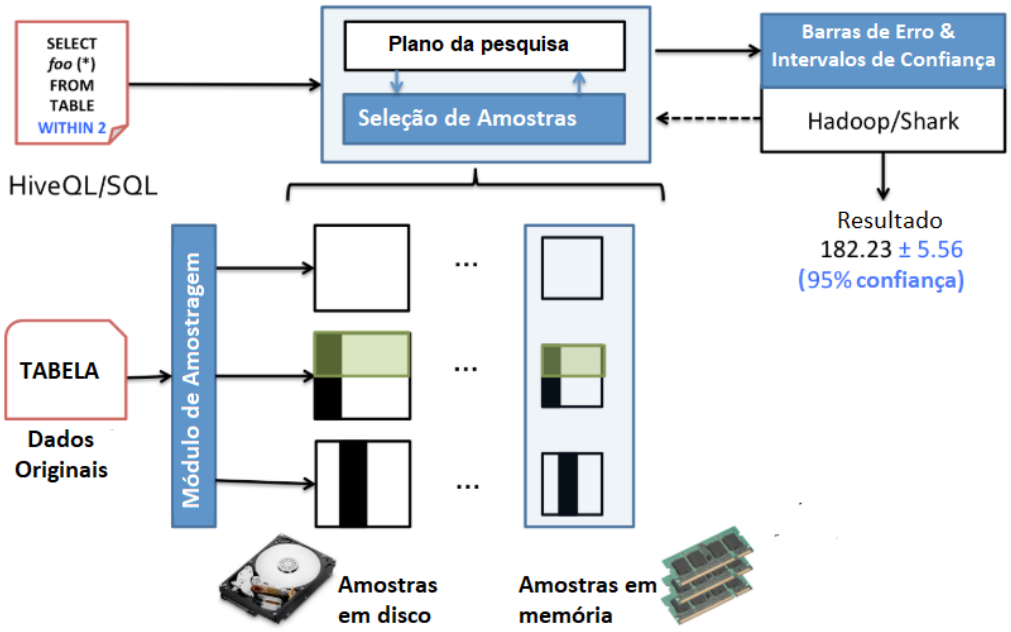

Figura 3.3: Modo de funcionamento do BlinkDB - Adaptado de [2]

A Tabela 3.3 apresenta as principais vantagens e desvantagens do BlinkDB. A amostra-

gem estratificada origina elevados custos de armazenamento no BlinkDB, sendo possível construir apenas um número limitado de amostras. Para diminuir esses custos, são apenas selecionadas algumas das tabelas para amostrar. Todas as amostras armazenadas em *cache* originam muitos planos de execução diferentes para a mesma pesquisa. O BlinkDB seleciona o melhor plano de execução para cada pesquisa através da execução de cada plano em cada sub-amostra de dados. O BlinkDB utiliza as colunas que são frequentemente usadas para realizar a amostragem sobre elas e, em seguida as amostras são armazenadas em *cache* para que as próximas pesquisas sejam respondidas através das anteriores. Assim, não é possível processar pesquisas sobre novos dados, onde as pesquisas/dados anteriores não são conhecidos. Como são processados dados em lote durante muito tempo, o BlinkDB não é uma boa solução para análises em tempo real.

Tabela 3.3: Vantagens e desvantagens do BlinkDB

Vantagens	**Desvantagens**
- Para diminuir os custos de armazenamento, são apenas selecionadas algumas colunas para amostrar. - Seleciona o melhor plano de execução para cada pesquisa.	- Não consegue processar pesquisas sobre novos dados, onde as pesquisas/dados geralmente não são conhecidos. - Processa dados em lote e não fornece uma baixa latência para análises em tempo real.

3.4 ApproxHadoop

O *ApproxHadoop* [26] é um sistema que foi desenvolvido para criar e executar programas que suportem uma aproximação de MapReduce. Este sistema possui mecanismos de aproximação que combinam com o modelo do MapReduce, sendo eles:

1. **Amostragem dos dados de entrada**
   A amostragem dos dados de entrada foi implementada em novas classes, que analisam um bloco de dados de entrada e devolvem uma amostra aleatória dos dados de entrada de acordo com uma determinada taxa de amostragem, definida pelo utilizador. Assim, apenas será processado um subconjunto dos dados de entrada.
2. ***Task dropping***
   O *JobTracker*, que comunica com o *NameNode* para determinar qual a localização dos dados de cada *job*, foi modificado de modo a executar tarefas *maps* de forma aleatória, para observar os requisitos da amostragem em várias etapas e de modo a ser capaz de cancelar os *maps* em execução e eliminar os que estão pendentes. A classe *Reducer* também foi modificada, para conseguir detetar as tarefas de *map* que foram eliminadas, possibilitando que continue a sua execução sem esperar por resultados. Assim, apenas será executado um subconjunto das tarefas.
3. **Aceitação e execução de uma versão aproximada**
   O utilizador inicialmente envia o *job*, especificando um limite de erro e um intervalo de confiança. O *JobTracker* inicia várias tarefas de *map*, que posteriormente são reduzidas. Cada tarefa do *map* é executada através da amostragem do seu bloco de dados de entrada. O *map* termina a sua tarefa e devolve as suas estatísticas para a tarefa de redução. Essa tarefa estima os limites de erro e recorre ao *JobTracker* para terminar as restantes tarefas de *map*. O *JobTracker* cancela as tarefas de *map*, que ainda possam estar em execução. Em seguida a tarefa de redução é informada de que

a tarefa de *map* não será concluída, permitindo que seja concluída a sua conclusão e sejam gerados resultados.

Uma amostragem de duas etapas é aplicada no *MapReduce*, associando a população, os *clusters* e o valor associado a cada unidade na população aos respetivos componentes de *MapReduce*. A população corresponde aos dados de entrada, onde cada dado é uma unidade; cada bloco de dados resultantes da amostra é um *cluster*. Com esta associação, é possível obter um cálculo aproximado através da execução de apenas um subconjunto escolhido aleatoriamente das tarefas do *map*.
A Figura 3.4 mostra uma possível execução aproximada do programa ApproxWordCount, que efetua uma contagem aproximada de palavras, através da amostragem de várias etapas no ApproxHadoop. O utilizador inicialmente envia o *job* especificando um limite de erro e um intervalo de confiança. O *JobTracker* inicia tarefas de *map* e reduz as tarefas. Cada tarefa do *map* é executada através da amostragem do bloco de dados de entrada. A tarefa do *map* quando termina reporta as suas estatísticas para a tarefa de *reduce*, que estima os limites de erro e indica ao *JobTracker* para terminar as restantes tarefas do *map*. O *JobTracker* termina essas tarefas, mesmo que ainda possam estar em execução. A tarefa de *reduce* é informada de que as tarefas não serão concluídas, permitindo assim que seja gerada uma resposta.

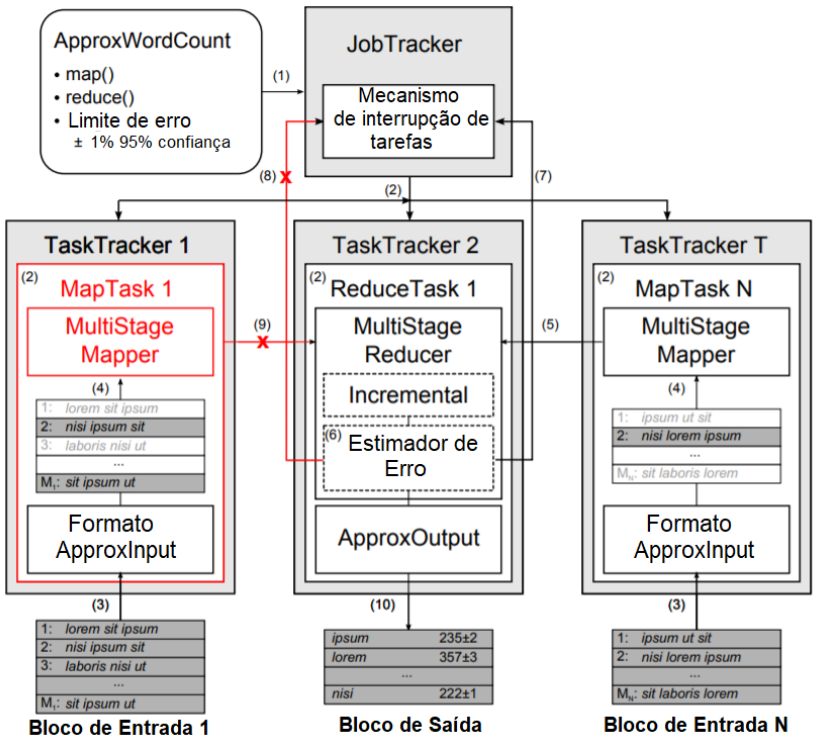

Figura 3.4: Exemplo de execução do programa ApproxWordCount com amostragem de várias etapas - Adaptado de [26]

A Tabela 3.4 apresenta as principais vantagens e desvantagens do ApproxHadoop. O utilizador pode especificar explicitamente qual a fração de tarefas de *map* que pretende descartar. Para não exceder os limites de erro, o ApproxHadoop realiza uma amostragem dos dados através de diversas etapas. Os métodos de amostragem e estimativa do ApproxHadoop funcionam bem supondo que os sub-conjuntos de dados são distribuídos uniformemente, no entanto, em muitos casos esses sub-conjuntos são distribuídos de forma não uniforme. Como o ApproxHadoop processa muitos dados em lote durante muito tempo,

não é indicado para analisar dados em tempo real.

Tabela 3.4: Vantagens e desvantagens do ApproxHadoop

Vantagens	Desvantagens
- O utilizador pode especificar qual a fração de tarefas de *map* que podem ser descartadas. - Efetua a amostragem dos dados em várias etapas.	- Considera que os sub-conjuntos de dados são distribuídos de forma uniforme em todo o conjunto de dados. - Processa dados em lote e não fornece uma baixa latência para análise em tempo real.

3.5 StreamApprox

Nos atuais sistemas de AQP os dados de entrada permanecem inalterados durante todo o processamento das pesquisas, o que implica que estes sistemas não sejam indicados para uma análise de *stream*. Isto foi o que motivou a criação do sistema StreamApprox [49], que utiliza uma análise de *stream* juntamente com o processamento aproximado de pesquisas. Este sistema utiliza uma amostragem estratificada para produzir uma resposta aproximada com limites de erro, sem requerer alterações internas na base de dados.
Para garantir que o fluxo de dados de entrada seja processado dentro do orçamento especificado, é utilizado um processamento aproximado, utilizando apenas um subconjunto dos itens de dados. Para determinar qual o subconjunto utilizado, o StreamApprox utiliza uma técnica de amostragem estratificada para selecionar e processar os itens de dados, onde o tamanho da amostra é determinado com base no orçamento definido pelo utilizador. A análise dos dados em tempo real é suportada pela técnica de Amostragem de Reservatório Estratificado Adaptativo Online (OASRS), que não ignora *sub-streams*, nem necessita de conhecer as suas estatísticas antes do processo de amostragem, funcionando forma eficiente em tempo real.
A Figura 3.5 apresenta a arquitetura de alto nível do StreamApprox. O StreamApprox recebe dados de diversas fontes, onde os itens de dados de cada fonte formam um sub-fluxo. Para combinar os itens de dados de entrada de cada sub-fluxo é utilizado um agregador de fluxo, por exemplo, o Apache Kafka. Depois dos sub-fluxos estarem agregados, um fluxo combinado é utilizado como entrada para a análise de dados. O utilizador especifica qual a pesquisa que pretende executar em *streaming* e o seu orçamento. Esse orçamento pode ser representado através de requisitos de latência, taxa de transferência esperada ou através do nível de precisão dos resultados.

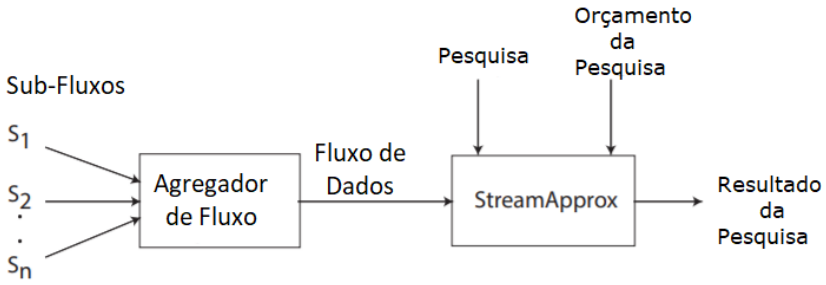

Figura 3.5: Arquitetura de alto nível do StreamApprox - Adaptado de [49]

A Tabela 3.5 apresenta as principais vantagens e desvantagens do StreamApprox. O StreamApprox é genérico e pode ser aplicado a dois métodos de processamento em tempo real: em lote e através da segmentação de instruções. Os dados de cada *substream* podem não seguir a mesma distribuição, no entanto, o StreamApprox assume que os dados seguem a mesma distribuição e são independentes.

Tabela 3.5: Vantagens e desvantagens do StreamApprox

Vantagens	**Desvantagens**
- Pode ser aplicado a dois modelos de processamento de tempo real.	- Assume que os itens de cada conjunto de dados seguem a mesma distribuição e são independentes.

3.6 AQP++

Existem dois métodos principais de AQP: o teorema do limite central e o método de *bootstrap*. O primeiro recolhe de uma amostra aleatória um conjunto mais pequeno do conjunto total de dados. Em seguida, são calculados a estimativa da amostra e o seu intervalo de confiança com base na teoria do limite central. Caso a precisão obtida seja satisfatória, é devolvido ao utilizador o resultado aproximado. O teorema do limite central é muito eficiente, mas apenas é adequado para pesquisas de agregação simples. Para uma determinada amostra aleatória simples, o *bootstrap* forma repetidamente conjuntos de dados através da reamostragem de linhas e em seguida calcula a pesquisa em cada um dos conjuntos de dados, no final calcula a estimativa média da amostra e avalia a qualidade do resultado. O método de *bootstrap* é muito geral, mas possui uma sobrecarga computacional elevada. Para tornar estes métodos mais gerais e eficientes, foi desenvolvida uma estrutura híbrida, designada AQP ++ [54], que combina as vantagens de ambos os métodos e elimina as suas limitações.
A Figura 3.6 apresenta o modo de funcionamento do AQP++, ilustrando os seus quatro componentes principais: (i) *CLT-based OLA* (ii) método *Bootstrap*, (iii) *full scan method* e (iv) *scheduling algorithm*. Dado um conjunto de pesquisas, inicialmente é calculado o custo de processamento através do teorema do limite central. Um modelo probabilístico é derivado para determinar o tamanho esperado da amostra. De acordo com o custo de processamento, o *scheduling algorithm* determina se irá escolher o *CLT-based OLA* para processamento. Se um limite de custo pré-definido for alcançado, a pesquisa será processada pelo método de *bootstrap* em vez do *CLT-based OLA*. Caso a pesquisa seja processada pelo método *bootstrap*, é necessário calcular o custo de processamento *bootstrap*. Em seguida o *scheduling algorithm* decide quando é que o processo de inicialização é terminado e alterna para o *full scan method* para obter os resultados. Finalmente, se a pesquisa for transferida transferido para o *full scan method*, significa que ocorreu uma falha de estimativa, sendo adotado o *full scan method* para devolver resultados precisos ao utilizador.

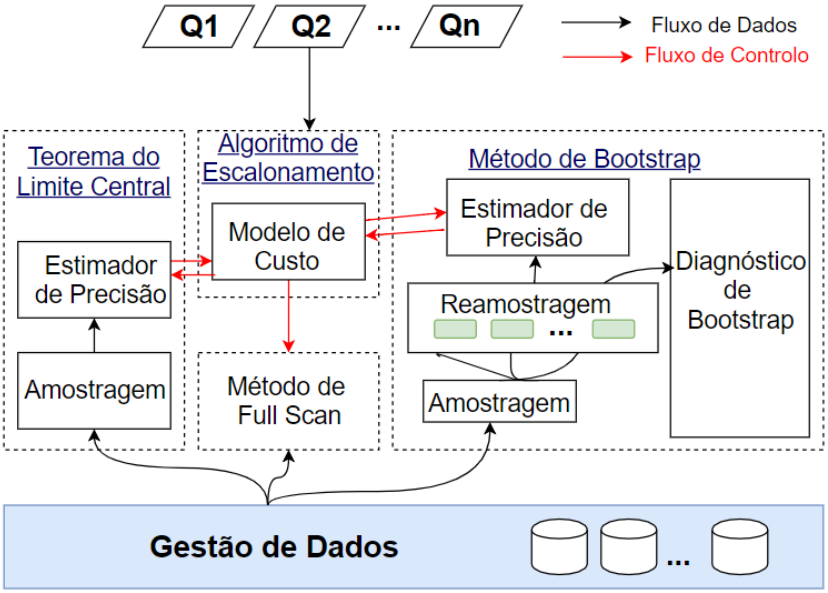

Figura 3.6: Funcionamento do AQP++ - Adaptado de [53]

A Tabela 3.6 apresenta as principais vantagens e desvantagens do AQP++. O AQP++ pode ser estendido para suportar outras técnicas de otimização, como a amostragem estratificada, permitindo que o seu desempenho seja melhorado. No AQP++ são utilizadas respostas exatas pré-calculadas sobre os dados completos em vez de respostas aproximadas sobre amostras para calcular as pesquisas futuras, garantindo que estas sejam melhoradas. O AQP++ não fornece resultados aproximados de forma rápida quando são utilizadas algumas funções de agregação, como MIN e MAX.

Tabela 3.6: Vantagens e desvantagens do AQP++

Vantagens	Desvantagens
- Pode ser ser facilmente estendido para suportar outras técnicas de otimização. - Melhora as pesquisas futuras através de respostas exatas pré-calculadas.	- Não funciona bem para algumas funções de agregação.

3.7 Intelli

A resposta a uma pesquisa revela sempre algum conhecimento difuso sobre a resposta a outra pesquisa, mesmo que as duas pesquisas acedam a diferentes linhas e colunas. No seguimento desta ideia, surgiu o *Database Learning* [45] que tem um comportamento muito semelhante a vários agentes de aprendizagem de inteligência artificial. O modelo interno do *Database Learning* captura as caraterísticas dos dados mais prováveis, de acordo com os pares observados da pesquisa e da sua resposta aproximada. Este mecanismo utiliza uma amostragem aleatória, juntamente com uma função de distribuição de probabilidades conjunta sobre as respostas da pesquisa, determinada utilizando o princípio de entropia máxima [29]. Este princípio determina qual a explicação mais provável para os valores agregados, de acordo com a quantidade de conhecimento extraída de pesquisas anteriores.

De acordo com o *Database Learning*, os autores criaram um mecanismo de AQP sobre o Spark SQL, chamado Intelli. Este mecanismo possui uma sinopse de consulta, um modelo e três módulos de processamento: (i) um mecanismo de processamento de AQP, (ii) um módulo de inferência e (iii) um módulo de aprendizagem. A sinopse de consulta possui um resumo das pesquisas executadas anteriormente e das suas respostas aproximadas. Quando a penúltima pesquisa é executada, é adicionada à sinopse, juntamente com a sua resposta aproximada e o erro estimado. O modelo representa o entendimento estatístico sobre os dados subjacentes, sendo treinado durante a sinopse de consulta e atualizado sempre que é adicionada uma nova pesquisa. Quando é executada uma pesquisa, é chamado o mecanismo de AQP para calcular uma resposta aproximada e um erro estimado. Em seguida, essa resposta aproximada é combinada com o modelo previamente treinado, de modo a inferir melhores respostas e estimativas de erro.
As pesquisas complexas são decompostas em pesquisas mais simples. A resposta para cada pesquisa mais simples é modelada de acordo com uma variável aleatória probabilística, correspondendo a uma integração sobre linhas relevantes extraídas de uma distribuição subjacente desconhecida. Para alcançar a generalidade dos dados, utilizaram um modelo probabilístico não paramétrico, segundo o qual as respostas a pesquisas mais simples que sejam diferentes são relacionadas através de uma função conjunta de distribuição de probabilidade (pdf). Derivaram essa função através do princípio da máxima entropia, que produz o pdf mais provável, de acordo com as queries anteriores e as suas respostas aproximadas. Finalmente, para garantir a eficiência computacional do sistema, restringiram-se apenas à média, variância e covariância ao aplicar o princípio da entropia máxima.
A Figura 3.7 descreve o modo de funcionamento do Intelli. Quando é realizada uma pesquisa, o módulo de inferência melhora a resposta obtida por uma pesquisa anterior, utilizando uma *query sinopsis* e um modelo. Sempre que uma pesquisa é processada, a uma resposta é adicionada à sinopse que posteriormente é utilizada pelo módulo de aprendizagem, para melhorar o modelo atual.

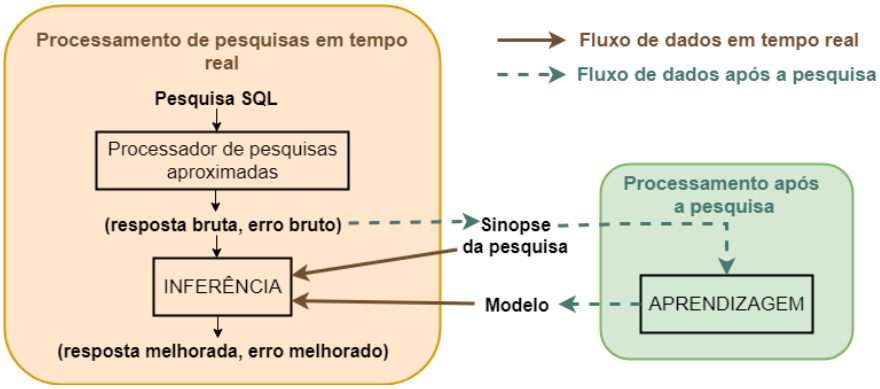

Figura 3.7: Modo de funcionamento do Intelli - Adaptado de [45]

A Tabela 3.7 apresenta as principais vantagens e desvantagens do Intelli. Contrariamente aos índices e vistas materializadas, o *database learning* incide em poucas sobrecarga de armazenamento, pois armazena apenas as pesquisas agregadas e as suas respostas. Consequentemente, os índices e vistas materializadas aumentam o seu tamanho sempre que aumenta a quantidade dados, enquanto que o *database learning* permanece igual com a ocorrência de alterações na quantidade dos dados. Como o *database learning* melhora o seu desempenho sempre que uma pesquisa é processada, o escasso processamento de pesquisas pode diminuir o seu desempenho, apresentando posteriormente resultados pouco precisos.

Tabela 3.7: Vantagens e desvantagens do Intelli

Vantagens	Desvantagens
- O *database learning* incide em pouca sobrecarga de armazenamento. - O modelo é treinado sempre que é adicionada uma nova pesquisa.	- Se forem processadas poucas pesquisas, o Intelli pode obter resultados com pouca precisão. - Não suporta sub-pesquisas.

3.8 VerdictDB

O *VerdictDB* [44] é um sistema de AQP que utiliza uma arquitetura de *middleware* que não requer alterações na base de dados, sendo possível funcionar com todos os sistemas de gestão de dados disponíveis no mercado. O utilizador envia as pesquisas para o *VerdictDB* e obtém o resultado das mesmas sem interagir com a base de dados. Em seguida, o *VerdictDB* comunica com a base de dados para obter metadados e para aceder e processar dados. Se existir um conjunto de tabelas da amostra que possa ser utilizado no lugar de cada uma das tabelas base, são utilizadas as tabelas de amostra na pesquisa. Cada pesquisa é convertida em operadores lógicos e, em seguida é convertida para outra expressão lógica capaz de ser executada no AQP, no final é reescrita numa instrução SQL, para ser executada.
Durante a preparação da amostra, o *VerdictDB* cria tabelas de amostra de acordo com um dos três tipos de amostragem: (i) uniforme, (ii) com hash e (iii) estratificada.
A subamostragem é utilizada como técnica para estimar o erro, através da subamostragem variacional, que reduz o custo da subamostragem tradicional. A subamostragem variacional é mais eficiente, uma vez que a mesma linha da tabela pode pertencer a várias subamostras e todas as subamostras devem ter o mesmo tamanho. Para cada linha da subamostra é gerado apenas um número aleatório para determinar a qual subamostra pertence e é executada em seguida a agregação apenas uma vez por linha. O utilizador define quais as tabelas sobre as quais pretende construir amostras, sendo normalmente escolhidas as que contêm um maior conjunto de dados.
A Figura 3.8 apresenta a arquitetura interna do VerdictDB. Quando é realizada uma pesquisa, o *Query Parser* converte-a em expressões lógicas como por exemplo em junções, que em seguida são convertidas pelo *AQP Rewriter* em outra expressão lógica capaz de ser executada com AQP. O *Syntax Changer* converte essa expressão lógica reescrita numa instrução SQL que pode ser executada na base de dados subjacente. No final, depois dessa pesquisa reescrita ser executada, o *Answer Rewriter* devolve uma resposta aproximada à pesquisa original.

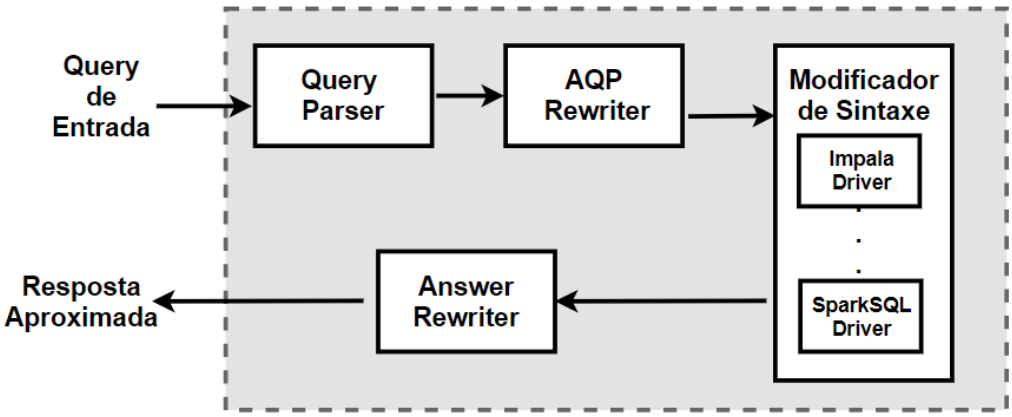

Figura 3.8: Arquitetura interna do VerdictDB - Adaptado de [44]

A Tabela 3.8 apresenta as principais vantagens e desvantagens do VerdictDB. O VerdictDB utiliza uma arquitetura de *middleware*, que não requer alterações na base de dados. Este permite que o utilizador selecione quais as tabelas que pretende amostrar, possibilitando que este escolha as tabelas ideais para amostrar (normalmente são as tabelas que possuem mais dados). O processo de construção de uma tabela amostrada é lento, chegando a demorar quatro horas para uma tabela com 35GB de dados.

Tabela 3.8: Vantagens e desvantagens do VerdictDB

Vantagens	**Desvantagens**
- Não requer alterações na base de dados. - É possível selecionar quais as tabelas a amostrar.	- O processo de construção de uma tabela amostrada é muito lento.

3.9 BigIN4

Para permitir a identificação interativa de *insights*, ou seja, para identificar as ações tangíveis que podem ser usadas para melhorar um negócio, é necessário procurar um grande número de combinações de dimensões e uma série de pesquisas de agregação necessitam de ser respondidas rapidamente. Para análise de big data em *business inteligence*, a identificação de *insights* é um processo que consome muitos recursos. Para contornar esse problema, surge o *BigIN4* [37], que permite identificar de forma instantânea e interativa *insights* a partir de grandes volumes de dados multi-dimensionais. Um *insight* derivado de um conjunto de dados multidimensional está associado a um sub-espaço. Para cada sub-espaço definido pelo utilizador, o *BigIN4* enumera-os de cima para baixo, começando pelo sub-espaço definido e adiciona novas dimensões e valores até que uma profundidade definida pelo utilizador seja atingida. Os *insights* são identificados através do módulo de processamento de pesquisas em cada iteração.
O *BigIN4* utiliza um novo algoritmo de AQP, o DynamicBayes [25], que decompõe as pesquisas que não são possíveis de obter uma resposta, em várias pesquisas de baixa dimensão de modo a estimar a resposta através de uma rede *Bayesiana*. O *DynamicBayes* possui três etapas: inicialmente é construída uma matriz para descrever a relação de dependência entre as dimensões da pesquisa; em seguida é criada uma árvore de amplitude máxima

como o esqueleto da rede e, por fim são adicionadas mais arestas de dependência à rede, desde que todas as pesquisas possam ser respondidas.
A Figura 3.9 ilustra o modo de funcionamento do BigIN4. Quando é realizada uma pesquisa que utiliza uma grande dimensão de dados, o BigIN4 decompõe essa pesquisa de alta dimensão em várias pesquisas de baixa dimensão. Posteriormente, é utilizado um cubo de dados para responder à pesquisa e, se não for encontrado nenhum cubo é chamado o *DynamicBayes* para obter uma resposta aproximada dos cubos de dados.

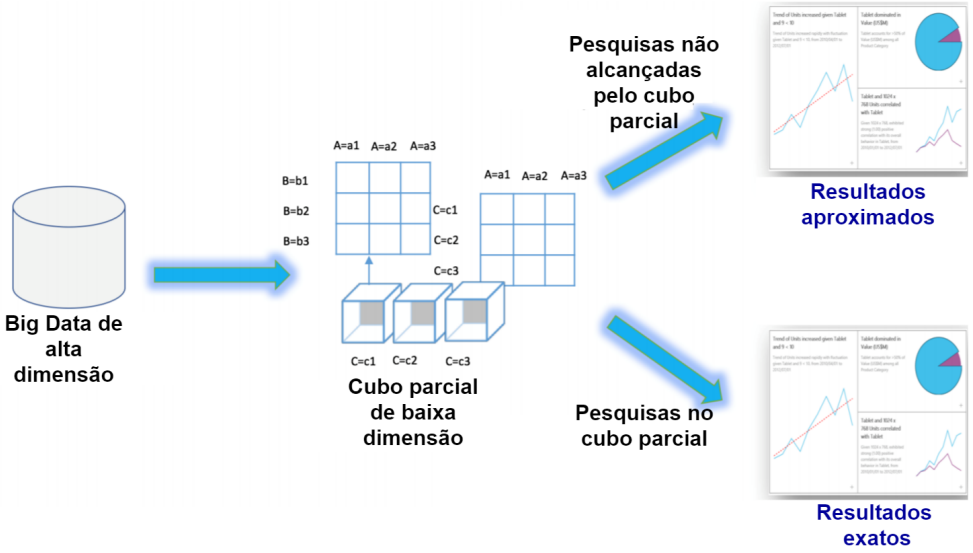

Figura 3.9: Modo de funcionamento do BigIN4 - Adaptado de [37]

A Tabela 3.9 apresenta as principais vantagens e desvantagens do BigIN4. O BigIN4 difere-se dos restantes sistemas de AQP apresentados neste documento por dois motivos principais: (i) funciona para sistemas de BI e (ii) não possui limitações acerca da construção das pesquisas, possibilitando que seja utilizado qualquer tipo de pesquisa, incluindo pesquisas com funções de agregação e com junções. O BigIN4 possui uma desvantagem muito grande em relação aos sistemas anteriormente descritos, uma vez que suporta a agregação numa única tabela de factos, não suportando junções.

Tabela 3.9: Vantagens e desvantagens do BigIN4

Vantagens	Desvantagens
- Funciona para sistemas de BI. - Não possui limitações acerca da construção das pesquisas.	- Apenas suporta a agregação de pesquisas numa única tabela de factos.

3.10 Comparação dos sistemas de processamento aproximado de pesquisas

A Tabela 3.10 fornece uma comparação entre os sistemas de AQP. Esta comparação incide nos modos de processamento destes sistemas (Online/Offline), na caraterização do sistema (Standalone/Distribuído), no algoritmo base que utilizam e nas plataformas em que operam.

Tabela 3.10: Comparação dos sistemas de AQP

	Processamento	**Sistema**	**Algoritmo**	**Plataforma**
ApproxHadoop	Online	Distribuído	Approximation-enabled MapReduce	Hadoop
AQP++	Online	Standalone	CLT-based OLA/Bootstrap	Visual Studio
ApproxJoin	Online	Distribuído	Amostragem Estratificada/Bloom filters	Spark
BigIN4	Offline	Distribuído	DynamicBayes	Hadoop/Spark
BlinkDB	Online	Distribuído	Amostragem Estratificada	Hive/Hadoop
Intelli	Offline	Distribuído	Database Learning	Spark
StreamApprox	Online	Distribuído	Adaptive Stratified Reservoir Sampling	Spark Streaming/Flink
VerdictDB	Offline	Distribuído	Subamostragem Variacional	MySQL/Spark/PostgreSQL
XDB	Online	Standalone	Wander Join	PostgreSQL

Os sistemas de AQP online não necessitam de prever inicialmente as latências das pesquisas. Em vez disso retornam as respostas à medida que são processados mais dados ou lidas mais pesquisas e param quando o utilizador está satisfeito ou o orçamento de tempo foi esgotado. Os sistemas de AQP offline necessitam de selecionar um tamanho apropriado para a amostra, sem conhecerem requisitos de latência.
Os sistemas atuais de AQP apresentam diversos problemas. A maioria destes sistemas exigem que seja realizada alguma modificação nos mecanismos de base de dados existentes, como é o caso do XDB que foi implementado diretamente no PostgreSQL. Todos os sistemas apresentados na Tabela 3.10 devolvem estimativas de erro que são incompatíveis com as ferramentas de BI. Com esta incompatibilidade, a maioria dos utilizadores recusa-se a utilizar sistemas de AQP se estes não forem compatíveis com as ferramentas de BI que eles atualmente utilizam e se não puderem ser utilizados como um substituto da DW. Uma possível solução para resolver este problema seria projetar uma arquitetura de AQP baseada em *middleware*, que não dependesse de nenhuma modificação interna no mecanismo de base de dados, como é o caso do VerdictDB. A maioria dos sistemas de AQP não são portáveis, sendo desenhados para serem utilizados em sistemas operativos específicos, como é o caso do UBUNTU. O último problema relaciona-se com o *design* de interfaces do utilizador, que continua a ser uma área aberta de pesquisa, uma vez que muitos utilizadores consideram difícil interpretar o erro de uma resposta aproximada [39], uma vez que este só é conhecido depois da pesquisa completa ser executada.

Capítulo 4

Abordagem Proposta: JDBCApprox

Este capítulo apresenta a abordagem JDBCApprox, proposta para colmatar as seguintes deficiências dos atuais sistemas de AQP: (i) não requer que seja efetuada qualquer alteração na base de dados, possuindo uma arquitetura de *middleware*; (ii) possuir uma interface, que permite que os utilizadores compreendam facilmente os resultados devolvidos; (iii) é possível parametrizar o grau de confiança e o erro máximo admitido e (iv) lida com pesquisas que utilizem mais do que uma vez a mesma tabela. Esta abordagem visa o processamento de amostras em memória para acelerar as respostas às pesquisas, sendo projetada para ser modular e flexível, de modo a que seja possível adicionar novas funcionalidades. Inicialmente são apresentados os motivos que estiveram na origem desta abordagem. Em seguida é explicada qual a técnica de amostragem aplicada e qual a arquitetura do JBCApprox. Posteriormente são abordadas caraterísticas importantes da abordagem proposta, como a atualização dos dados, gestão de memória e o tempo de execução das pesquisas. Por último é apresentada a interface do JDBCApprox.

4.1 Motivação

Uma grande parte dos atuais sistemas de AQP modificam o algoritmo de junção [35], fazem alterações no motor de base de dados [35] ou no plano da pesquisa [43]. Os sistemas de AQP que possuem uma arquitetura de *middleware*, como o caso do VerdictDB [44], são escassos. O VerdictDB possui três limitações: (i) não suporta todo o tipo de pesquisas; (ii) não é possível parametrizar e obter o grau de confiança e o erro máximo admitido. O tipo de pesquisas que não são suportadas por este são as que referenciam uma tabela mais do que uma vez, por exemplo numa sub-pesquisa, sendo estas pesquisas mais lentas. O facto deste não possuir uma interface para os utilizadores, torna-se difícil a sua utilização e a compreensão dos resultados obtidos. Além disso, como não é possível parametrizar o grau de confiança e o erro máximo admitido, torna-se incapaz de compreender o quanto exata é a resposta a uma determinada pesquisa. A abordagem proposta tem como objetivo eliminar essas limitações e as deficiências dos atuais sistemas de AQP, através do desenvolvimento de uma biblioteca Java que utiliza uma amostragem aleatória simples sem repetição para criar amostras das tabelas e, em seguida recorre a uma base de dados em memória para responder de forma rápida às pesquisas.

4.2 Técnica de amostragem

A amostragem foi utilizada no JDBCApprox, uma vez que as abordagens baseadas em histogramas e *wavelets* assumem que os atributos das tabelas são dados numéricos [24]. As abordagens baseadas em amostragem não exigem tais suposições. Além disso, apenas as abordagens baseadas em amostragem foram exploradas para responder aproximadamente

a pesquisas agregadas utilizando junções de relações. De acordo com os tipos de amostragem apresentados no capítulo 2, a amostragem de reservatório e a amostragem aleatória simples foram previamente testadas como técnica para a construção de sinopses dos dados. Através da utilização da amostragem de reservatório os resultados aproximados das pesquisas eram muito diferentes dos resultados originais, apresentando um erro relativo percentual de cerca de 30% para pesquisas que utilizavam apenas uma junção. Esta elevada percentagem de erro tinha como origem diversos dados repetidos nas amostras que eram criadas. Posteriormente foi utilizada a amostragem aleatória simples sem repetição, uma vez que esta não seleciona mais do que uma vez o mesmo elemento para a amostra e é indicada para grandes conjuntos de dados, para permitir que sejam selecionados aleatoriamente mais elementos para a amostra. Na amostragem aleatória simples sem repetição [42] cada tabela é tratada de forma independente e no final as amostras resultantes de cada tabela são unidas. Os resultados finais através desta técnica originavam grandes percentagens de erro na estimativa do tamanho de uma junção, uma vez que não era garantido que a amostra de dados de uma tabela iria abranger todos os dados de uma outra amostra. Esta técnica de amostragem é benéfica quando se pretende obter uma resposta aproximada de uma pesquisa que recorre a apenas uma tabela.
A Figura 4.1 ilustra o exemplo da amostragem aleatória simples sem repetição aplicada a duas tabelas: Cliente e Encomendas, onde a percentagem de amostragem corresponde a 20% do total dos dados.

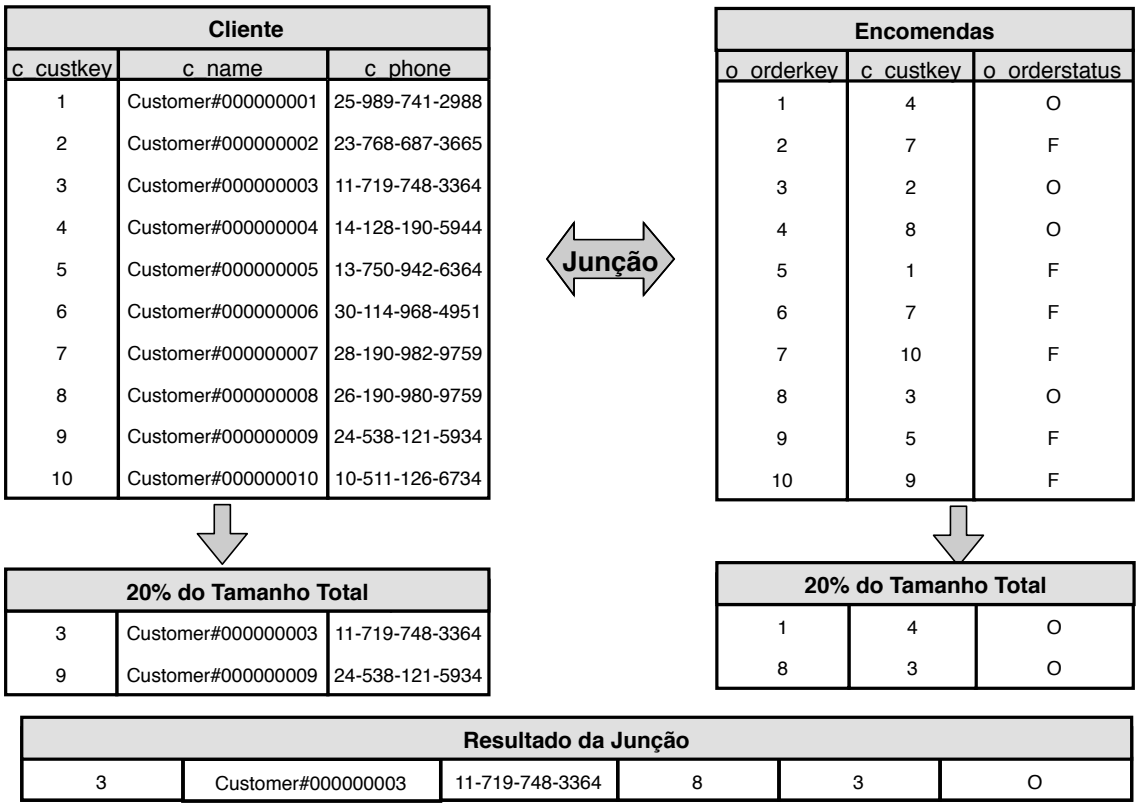

Figura 4.1: Amostragem aleatória simples sem repetição aplicada a duas tabelas

Selecionar aleatoriamente 20% dos dados de cada tabela resulta em duas linhas de cada tabela. Uma junção aplicada a essas duas linhas, baseada na chave estrangeira *c_custkey* resultaria em apenas uma linha. Para obter uma estimativa do tamanho da junção é necessário dividir o número de linhas da amostra pela percentagem da amostra. A estimativa resultante seria de cinco linhas e, caso a junção fosse aplicada no conjunto de dados origi-

nais resultaria em dez linhas. Assim, o erro relativo percentual seria de 50%.
De modo a reduzir o erro relativo percentual em pesquisas que recorrem a várias tabelas, foi projetada uma nova estratégia de amostragem. A ideia base é aplicar uma amostragem aleatória simples sem repetição à tabela presente na base de dados que possui mais chaves estrangeiras e, em seguida selecionar as linhas das restantes tabelas que se unem a essa amostra.
O número de dados selecionados da tabela que possui mais chaves estrangeiras é determinado através da seguinte equação [8]:

$$n = \frac{N \times Z^2 \times p \times (1-p)}{(N-1) \times e^2 + Z^2 \times p \times (1-p)} \tag{4.1}$$

Onde:

n = Número de dados da amostra a calcular.
N = Número de linhas da tabela sobre a qual se pretende calcular a amostra.
Z = Desvio do valor médio para alcançar o grau de confiança desejado. Em função do grau de confiança pretendido, é utilizado um valor que é dado pela forma da distribuição de Gauss [4] Os valores mais frequentes para **Z** são apresentados na Tabela 4.1.

Tabela 4.1: Valores de Z associados ao grau de confiança da amostra

Grau de Confiança	**Z**
90%	1.645
95%	1.96
99%	2.575

e = Percentagem de erro máximo que se pretende admitir.
p = Proporção de dados que não pertencem à categoria a estudar.
A proporção p aparece na fórmula, pois quando uma população é muito uniforme, a convergência para uma população normal é mais precisa e permite reduzir o tamanho da amostra. Como não é possível identificar qual a proporção de dados para criar uma amostra de uma tabela, é utilizado o seguinte cenário: a população distribui-se em partes iguais, logo $p = 50\%$.

Depois da quantidade de dados ser determinada, a amostra é criada selecionando dados de forma aleatória sem repetição, até perfazer essa quantidade. Em seguida, são identificadas as relações existentes com as restantes tabelas e as amostras são criadas de acordo com a seguinte regra: caso a tabela sobre a qual se pretende criar uma amostra (Tabela A) possua apenas uma chave estrangeira (Tabela B), a amostra é criada utilizando uma junção com a tabela que possui essa chave estrangeira (junção entre as tabelas A e B). Se a tabela (Tabela A) possuir diversas chaves estrangeiras, a amostra é criada utilizando uma junção com a tabela que contém um maior número de registos e possui essa chave estrangeira.

4.3 Arquitetura

A Figura 4.2 ilustra a arquitetura do JDBCApprox. O JDBCApprox é acedido através de código Java, que utiliza a sua API. Esta tem como função lidar com as pesquisas, através da sua divisão e reescrita, enviando para o gestor de amostras o nome das tabelas que a pesquisa possui. Em seguida, o gestor de amostras verifica se as amostras existem em memória e, caso existam a pesquisa reescrita é executada. Caso contrário, o gestor de amostras acede às informações presentes no ficheiro de configuração para criar amostras e proceder à cópias destas para memória.

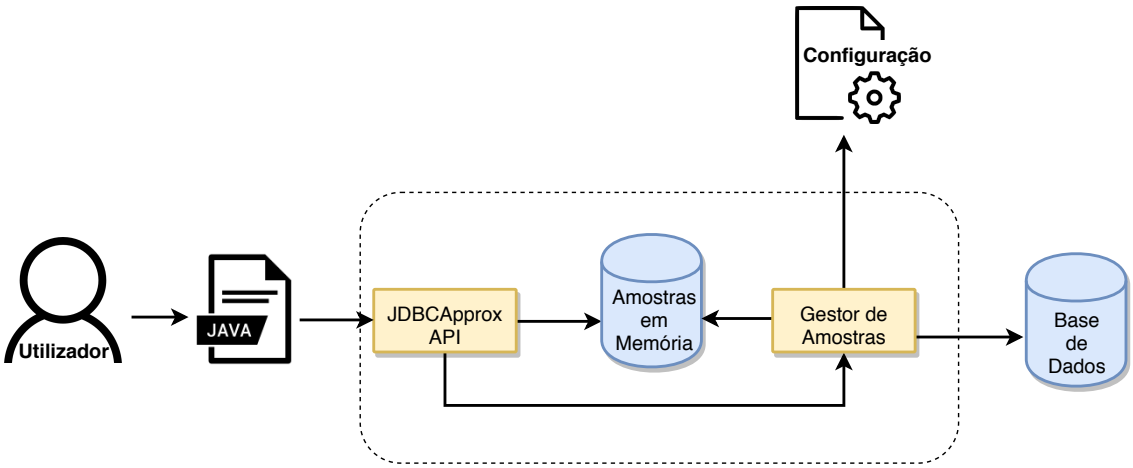

Figura 4.2: Arquitetura do JDBCApprox

O modo de funcionamento do JDBCApprox é detalhado em seguida.

Divisão da pesquisa

Inicialmente, quando o utilizador escreve uma pesquisa, esta é dividida. Essa divisão consiste em separar a pesquisa através das suas cláusulas (*SELECT*, *FROM*, *WHERE*, *GROUP BY*, *ORDER BY*), sendo realizada através da utilização de uma expressão regular. Do resultado desta expressão regular são extraídos os nomes das tabelas presentes na cláusula *FROM*. Outra expressão regular também é aplicada à pesquisa de modo a identificar quais as funções de agregação utilizadas e qual a sua posição na pesquisa, para posteriormente esta ser reescrita. Em seguida, os nomes das tabelas são enviados para o gestor de amostras, que verifica se as amostras dessas tabelas estão presentes na base de dados com configuração em memória.

Gestão das amostras

Depois de verificada a existência das amostras em memória existem dois cenários possíveis: (i) as amostras não estão presentes em memória e (ii) as amostras estão presentes em memória. No primeiro cenário é consultada a meta-informação das tabelas que estão presentes no PostgreSQL, de modo a identificar qual a tabela que possui um maior número de chaves estrangeiras e quais as chaves estrangeiras das restantes tabelas. Em seguida, o ficheiro de configuração é acedido para determinar qual o tamanho da amostra, de acordo com a equação 4.1, através da indicação do grau de confiança e da percentagem de erro máximo que se pretende admitir. Quando é determinado o tamanho da amostra, esta é criada de forma aleatória sem repetição, com um nome semelhante ao da tabela original. Por exemplo, se pretendemos obter uma amostra da tabela *lineitem*, esta irá possuir o nome *lineitem_sample*. Em seguida, as restantes amostras são criadas de acordo com a seguinte regra: caso a tabela sobre a qual se pretende criar uma amostra possua apenas uma chave estrangeira, a amostra é criada utilizando uma junção com a tabela que possui essa chave estrangeira. No segundo cenário, quando as amostras estão presentes em memória não é realizada nenhuma alteração nas mesmas.

Cópia das amostras para memória

Depois de criadas as amostras, é necessário que estas sejam copiadas para memória de modo a acelerar a resposta às pesquisas. Assim, as amostras presentes no PostgreSQL são copiadas para uma base de dados com configuração em memória, através de uma solução designada *pg2sqlite* [7]. Esta utiliza um *dump* das tabelas da base de dados, de modo a copiar os dados de forma rápida, alterando a estrutura das tabelas do PostgreSQL para serem criadas no SQLite.
Para copiar uma tabela para memória, neste caso a tabela *lineitem*, é utilizado o seguinte *script*:

```
#!/bin/sh
pg_dump -h 10.17.1.14 --table=tpch1g.lineitem_sample postgres > lineitem.dump
sed -i 's/tpch1g.//'lineitem.dump
sed -i 's/_sample.//'lineitem.dump
sed -i 's/timestamp without time zone./TEXT,/'lineitem.dump
sed -i 's/<00:00:00>/&.001/g'lineitem.dump
java -jar pg2sqlite.jar -d lineitem.dump -o lineitem.db
```

Neste *script* inicialmente é gerado o *dump* da amostra para o ficheiro *lineitem.dump*. Quando esse ficheiro estiver criado, são eliminados nomes da tabela, de modo a que esta fique com o mesmo nome do que a tabela original. O SQLite não possui uma classe de armazenamento reservada para armazenar dados como data/hora, sendo necessário converter as datas da amostra para texto. No final, é utilizado o *pg2sqlite* para criar uma base de dados SQLite através do *dump* modificado.

Reescrita da pesquisa

Depois de copiadas as amostras para memória, a pesquisa necessita de ser reescrita, de modo a que quando esta for executada produza um resultado estimado a partir das amostras. Para estimar o resultado de uma pesquisa divide-se o resultado das funções de agregação SUM ou COUNT pela percentagem da amostra, obtida a partir da equação 4.2.

$$Percentagem\,da\,Amostra = \frac{Tamanho\ da\ maior\ tabela}{Tamanho\ da\ amostra} \tag{4.2}$$

A figura 4.3 ilustra o exemplo da reescrita da pesquisa Q1, quando são utilizadas amostras com 20% dos dados originais. Neste caso foi adicionada uma divisão por 0.2 para as funções de agregação SUM e COUNT.

Pesquisa Q1

```
SELECT
l_returnflag,
sum(l_quantity) as sum_qty,
avg(l_quantity) as avg_qty,
count(*) as count_order
FROM
lineitem
WHERE
l_shipdate <= date '1998-12-01'
GROUP BY
l_returnflag;
```

Pesquisa Q1 reescrita

```
SELECT
l_returnflag,
sum(l_quantity)/0.2 as sum_qty,
avg(l_quantity) as avg_qty,
count(*)/0.2 as count_order
FROM
lineitem
WHERE
l_shipdate <= date '1998-12-01'
GROUP BY
l_returnflag;
```

Figura 4.3: Exemplo da reescrita da pesquisa Q1

Execução da pesquisa

No final, depois da pesquisa ser reescrita procede-se à sua execução. Esta consiste em efetuar a pesquisa reescrita no SQLite com uma configuração em memória. Esta configuração foi utilizada para as pesquisas serem respondidas de forma mais rápida, uma vez que as bases de dados em memória são mais rápidas do que as bases de dados em disco. Nas bases de dados em memória o acesso ao disco é mais lento do que o acesso a memória e os algoritmos de otimização interna são mais simples e executam menos instruções. Assim, aceder aos dados em memória fornece um desempenho mais rápido do que se estes forem acedidos em disco.

4.4 Seleção das Tecnologias

Foi efetuado um estudo e análise das ferramentas existentes, de modo a selecionar as que mais se adequavam para suportar a implementação do JDBCApprox. Este estudo deu a conhecer as caraterísticas e o modo de funcionamento das ferramentas selecionadas. Os critérios de seleção consistiam em escolher as ferramentas mais adequadas às nossas necessidades, dando preferência às *open source*. Neste subcapítulo é apenas efetuado um breve resumo das conclusões obtidas aquando a análise das ferramentas.

Linguagem de desenvolvimento

Como linguagem de desenvolvimento do JDBCApprox foi selecionado o Java, uma vez que esta é orientada a objetos e é uma linguagem multiplataforma, possuindo assim a capacidade de execução em diferentes sistemas operativos. Além disso, o Java possui um *driver* que permite a ligação a diferentes bases de dados, permitindo executar instruções SQL.

Base de dados

A base de dados possui um papel muito importante neste projeto, uma vez que será o local onde serão armazenados todos os dados para responderem a pesquisa. Por esse motivo foram estudados os quatro melhores servidores de base de dados presentes no top 10 do ano de 2019 [12], sendo estes: *Oracle Database*; *MySQL*; *Microsoft SQL Server* e *PostgreSQL*. Foram analisadas caraterísticas como as linguagens de programação suportadas e o modo

de desempenho quando se utilizam grandes quantidades de dados. Nesta análise destacou-se o *PostgreSQL*, uma vez que este normalmente funciona melhor quando é utilizado em sistemas que executam pesquisas complexas.
Em seguida foram analisadas as três bases de dados em memória mais utilizadas em Java [6], sendo estas: *H2 Database*, *HSQLDB* e *SQLite*. Foram analisadas caraterísticas como as linguagens de programação suportadas, a facilidade de utilização e a possibilidade de importar dados do *PostgreSQL*. Nesta avaliação destacou-se o *SQLite*, uma vez que era o mais simples de utilizar e era possível importar os dados do *PostgreSQL* de forma rápida.

4.5 Atualização dos dados

As amostras dos dados são tabelas armazenadas em memória, independentes da base de dados original. Quando existem alterações significativas nas tabelas da base de dados original, é necessário atualizar as amostras para que os resultados das pesquisas sejam o mais exatos. A primeira solução projetada seria construir um *trigger* na base de dados original, sendo acionado sempre que ocorresse uma inserção ou eliminação de dados. Este *trigger* necessitava de calcular estatísticas acerca dos dados, por exemplo calcular a média de um determinado atributo e comparar esse valor com a média da amostra, se existisse uma diferença elevada, então a amostra era apagada e posteriormente era criada uma nova. No entanto, isto seria um processo que iria aumentar o tempo de criação de amostras, uma vez calcular um valor médio de um atributo de uma tabela com milhões de registos é um procedimento lento. Além disso, um dos requisitos do JDBCApprox era violado, uma vez que este tem que ser independente da base de dados original. A melhor solução a adotar para a atualização dos dados será da responsabilidade do administrador de sistemas. Este tem como função definir qual a periodicidade das atualizações, através da indicação dos dias e hora para as amostras existentes serem eliminadas e posteriormente serem criadas novas amostras. Essa periodicidade está relacionada com a quantidade de dados armazenados e com a utilização da base de dados, por exemplo, podem ocorrer atualizações noturnas ou em períodos de menor utilização, e durante essas atualizações a base de dados não está disponível para pesquisas.

4.6 Gestão de memória

O JDBCApprox armazena as amostras na memória RAM, em vez de as armazenar na base de dados presente em disco. Isto elimina as operações de entrada/saída e acelera exponencialmente o acesso aos dados, pois os dados armazenados na memória RAM ficam disponíveis instantaneamente, enquanto que os dados armazenados em disco são limitados pela rede e pelas velocidades do disco. Com este tipo de armazenamento é possível colocar quantidades massivas de dados em memória cache, permitindo tempos de resposta rápidos, auxiliando na obtenção de um bom desempenho. Para este desempenho não diminuir com a contínua utilização do JDBCApprox é necessário efetuar uma boa gestão de memória, de modo a que exista uma grande quantidade de memória disponível sem eliminar todas as amostras armazenadas em memória. Caso todas sejam eliminadas irá ser necessário proceder novamente à sua cópia para memória, sendo consumido mais tempo.
De acordo com o princípio da localidade temporal [11], os dados acedidos recentemente têm uma maior probabilidade de serem utilizados novamente, do que os dados que foram acedidos há mais tempo. Isto relaciona-se com o facto das variáveis de um programa tenderem a ser acedidas diversas vezes durante a execução de um programa, e as instruções utilizam comandos de repetição e sub-programas, sendo as instruções acedidas repetidamente. Assim, o sistema de memória tende a manter os dados e instruções recentemente acedidos no topo da hierarquia de memória. Com base neste princípio, a solução mais correta para a gestão de memória no JDBCApprox consiste em manter em memória apenas as últimas

amostras utilizadas. Ou seja, quando termina a execução de uma pesquisa, as amostras que foram utilizadas permanecem em memória, para serem utilizadas em pesquisas futuras. Manter em memória apenas as amostras de menos dimensão seria uma solução menos boa, uma vez que estas demoram pouco tempo a serem copiadas para memória.

4.7 Tempo de execução de pesquisas

O tempo de execução das pesquisas é importante, na medida em que o utilizador pode preferir uma resposta exata, caso a resposta aproximada seja devolvida após ter decorrido muito tempo. A solução para o tempo de execução das pesquisas no JDBCApprox é executar as pesquisas através de um processamento *multithreading*, onde a pesquisa seria dividida em blocos, sendo apresentado sempre um resultado aproximado no final da execução de cada *thread*. Através deste modo de processamento o utilizador é capaz de parar a execução da pesquisa quando estiver satisfeito com o seu resultado. Prever o tempo de execução de uma determinada pesquisa de acordo com a complexidade da mesma, não seria uma solução para o tempo de execução das pesquisas no JDBCApprox. Esta complexidade poderia ser calculada através do número de junções que a pesquisa possui, no entanto, o tempo de execução está também relacionado com os índices de cada tabela. Deste modo, não é possível garantir que as pesquisas que utilizem as mesmas tabelas apresentem tempos de resposta muito similares.

4.8 Tipo de pesquisas suportadas

O JDBCApprox suporta pesquisas com as seguintes caraterísticas:

1. **Funções de Agregação**
   Qualquer número de funções de agregação SUM, COUNT, AVG, MAX, MIN, STDDEV e VARIANCE podem aparecer na cláusula SELECT. Estas funções de agregação também podem utilizar atributos derivados.

2. **Agrupamentos**
   A cláusula GROUP BY é suportada por todos os atributos armazenados e derivados.

3. **Junções**
   As junções de chave estrangeira com qualquer número de tabelas são suportadas.

4. **Seleções**
   Suporta qualquer comparação de igualdade e desigualdade para atributos categóricos e numéricos, incluindo o operador IN. Além disso, suporta operadores lógicos, como comparações AND e OR com sub-pesquisas.

5. **Tabelas**
   Suporta tabelas derivadas ou tabelas base unidas através de uma junção. A tabela derivada pode ser uma instrução SELECT com ou sem funções agregadas.

6. **Outras cláusulas**
   Suporta as cláusulas GROUP BY, ORDER BY, LIMIT e HAVING.

4.9 Limitações

O JDBCApprox possui um mecanismo de amostragem aleatória simples sem repetição para responder aproximadamente às consultas. Uma amostra de uma determinada tabela possui um pequeno número de linhas selecionadas aleatoriamente de uma tabela original. Isso fornece respostas aproximadas que estão muito longe da resposta original num caso

específico. Este caso acontece quando se pretende uma resposta aproximada a uma pesquisa que que utiliza dados agrupados por um atributo único. Por exemplo, para a seguinte pesquisa:

```
SELECT
    c_custkey, c_name, sum(l_extendedprice)
FROM
    customer, orders, lineitem
WHERE
    c_custkey = o_custkey
    AND l_orderkey = o_orderkey
GROUP BY
    c_custkey,
    c_name;
```

Os atributos *c_ custkey* e *c_ name* são únicos, o que significa que existe apenas uma linha na tabela *customer* para cada um desses atributos. Quando uma amostra da tabela *customer* é criada não pode possuir todos os dados para os atributos presentes na pesquisa. Portanto, o JDBCApprox possui uma limitação para esse tipo de pesquisas.

4.10 Interface

Esta secção apresenta a API do JDBCApprox e exemplos de como esta pode ser utilizada, pelos programadores e pelos utilizadores.

4.10.1 API - Interface de programação da aplicação

A interface de programação do JDBCApprox permite que os programadores a integrem em outros sistemas. Esta interface apresenta uma estrutura e utilização semelhantes à do JDBC, reunindo um conjunto de classes escritas na linguagem Java, onde é possível a ligação através de um controlador específico da base de dados desejada. Assim, é possível executar instruções SQL de qualquer tipo de bases de dados relacionais.
A Figura 4.4 ilustra um exemplo de utilização da API do JDBCApprox.

```
Connection conn = DriverManager.getConnection("jdbc:postgresql://10.17.1.14:5432/postgres");
StatementApprox approx = new StatementApprox(conn);
String query= "SELECT\n" +
        "    sum(l_extendedprice * l_discount) as revenue\n" +
        "FROM\n" +
        "    lineitem\n" +
        "WHERE\n" +
        "    l_shipdate >= date '1994-01-01'\n" +
        "    AND l_shipdate < date '1994-01-01' + interval '1' year\n" +
        "    AND l_discount between 0.06 - 0.01 AND 0.06 + 0.01\n" +
        "    AND l_quantity < 24;";
ResultSet rs = approx.executeQuery(query);
while(rs.next()) {
    System.out.println(rs.getDouble(1));
}
conn.close();
```

Figura 4.4: Exemplo de utilização da API do JDBCApprox

A API possui utiliza um ficheiro de configuração, que contém qual o grau de confiança e a percentagem de erro máximo que se pretende admitir, no entanto estas configurações

podem ser realizadas diretamente na API. Um exemplo desse ficheiro é apresentado na Figura 4.5.

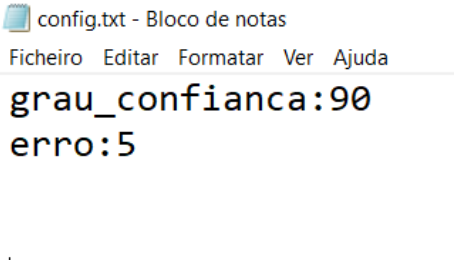

Figura 4.5: Ficheiro de configuração utilizado pela API

Para utilizar esta API inicialmente é necessário criar uma ligação entre a aplicação Java e uma base de dados específica, utilizando os parâmetros *url* (construído com informações importantes para estabelecer a ligação à base de dados: tipo de base de dados; endereço IP e número da porta onde a base de dados está armazenada), *user* (nome de utilizador a partir do qual a base de dados pode ser acedida) e *password* (palavra-passe a partir da qual a base de dados pode ser acedida). A ligação à base de dados é estabelecida através do seguinte código:

```
Connection conn = DriverManager.getConnection (url, user, password);
```

Depois de estabelecida a ligação, é possível interagir com a base de dados. Para esta interação se concretizar é necessário utilizar a instrução `StatementApprox`, que possui os métodos que permitem enviar instruções SQL e receber as informações da base de dados. Nesta instrução é estabelecida uma ligação com uma base de dados com configuração em memória. A interação com a base de dados é estabelecida através do seguinte código, onde `conn` é uma referência à interface de ligação utilizada na etapa anterior:

```
StatementApprox approx = new StatementApprox(conn);
```

Após ser utilizada a instrução `StatementApprox` as pesquisas SQL podem ser executadas. O método `executeQuery` é utilizado para executar pesquisas de recuperação dos valores da base de dados. Neste método a pesquisa é dividida, as amostras das tabelas utilizadas são criadas e a pesquisa é reescrita. Posteriormente é devolvido um objeto `ResultSet` que pode ser utilizado para obter todos os registos de uma tabela. Depois da pesquisa estar definida é possível obter o seu resultado aproximado através do seguinte código:

```
ResultSet rs = approx.executeQuery(query);
```

No final é necessário encerrar a ligação à base de dados. Quando esta é encerrada os objetos do `StatementApprox` e `ResultSet` são fechados automaticamente. O método `close` da interface `Connection` é utilizado para encerrar a ligação à base de dados:

```
conn.close();
```

4.10.2 GUI - Interface gráfica do utilizador

Um exemplo de aplicação da API é através da interface gráfica do utilizador. Esta é muito simples, de modo a seja possível uma rápida compreensão de todas as etapas necessárias

para conseguir executar uma pesquisa e ser devolvida uma resposta aproximada.
A figura 4.6 apresenta o primeiro ecrã que o JDBCApprox mostra quando a aplicação é iniciada. Este ecrã tem como finalidade estabelecer a ligação à base de dados e o utilizador deverá para esse efeito introduzir os dados necessários, ou simplesmente introduzir o URL de ligação à base de dados que é utilizado no código Java.

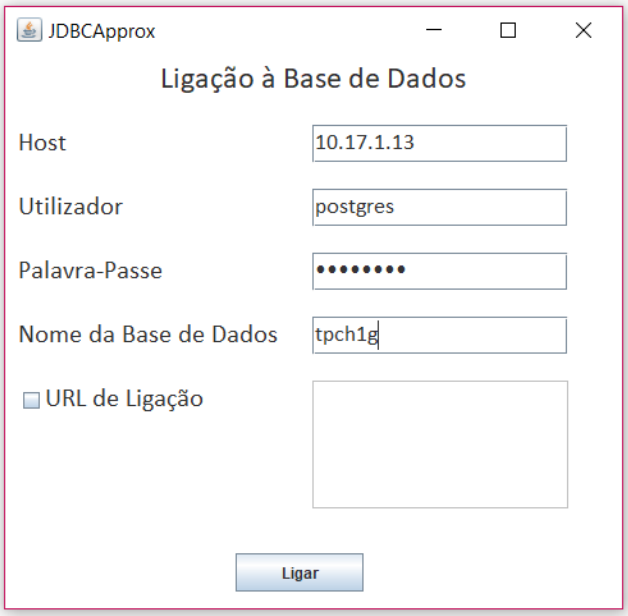

Figura 4.6: Interface da ligação à base de dados

Depois de estabelecida a ligação à base de dados, o utilizador pode optar por gerir as amostras na base de dados, como ilustra a figura 4.7. Através da meta-informação da base de dados é possível listar as tabelas existentes, existindo posteriormente as opções de criar amostras, ou apagar as que existem.

Figura 4.7: Interface de gestão de amostras

Quando são selecionadas as tabelas para posteriormente serem criadas amostras, é necessário indicar qual o grau de confiança pretendido e a percentagem de erro máximo que queremos aceitar, como ilustra a figura 4.8.

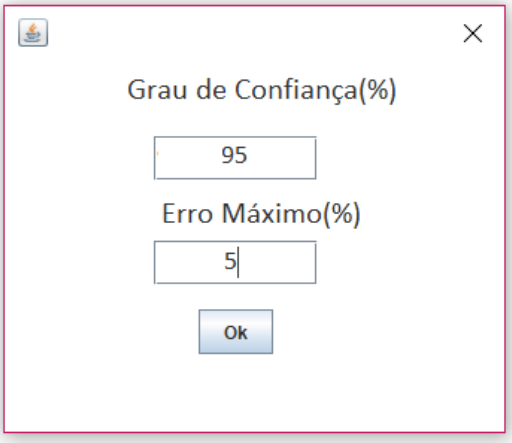

Figura 4.8: Interface de configuração das amostras

Após o passo de gestão de amostras, o utilizador introduz a pesquisa pretendida, sendo devolvidos em seguida a resposta e o tempo de execução. Este último passo está apresentado na figura 4.9.

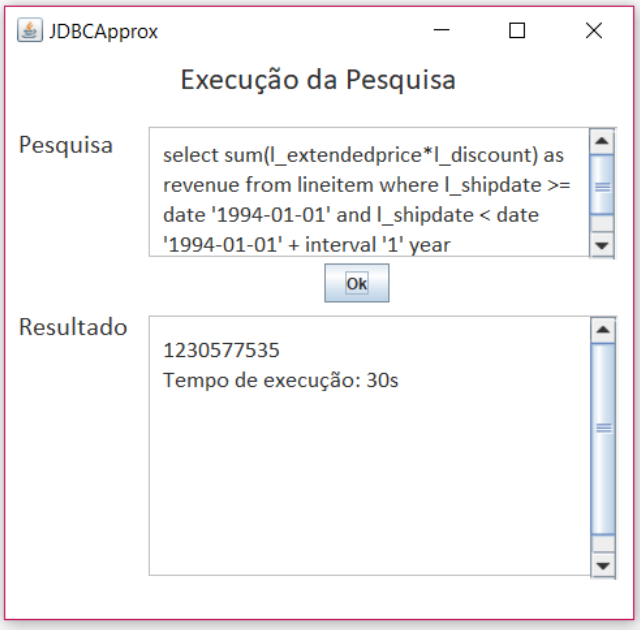

Figura 4.9: Interface de execução da pesquisa

Capítulo 5

Avaliação Experimental

No presente capítulo será realizada a avaliação experimental da abordagem proposta para respostas aproximadas em bases de dados. Inicialmente são apresentados os objetivos das experiências e o modo como será realizada a avaliação da abordagem proposta. Em seguida é apresentada uma descrição geral do ambiente experimental utilizado e das métricas utilizadas. A avaliação experimental será efetuada através de diferentes pesquisas à base de dados, que recorrem a diferentes tipos de funções de agregação e a diversos números de junções para o mesmo conjunto de dados. Nas experiências serão utilizados dois tamanhos distintos do conjunto de dados: 1GB e 10GB, sendo calculados o *speedup* e o erro apresentado na resposta de cada pesquisa. Na última secção é realizada uma análise dos resultados e são retiradas algumas conclusões.

5.1 Justificação das experiências

De modo a avaliar a abordagem proposta, foi selecionado o sistema de AQP que apresentava os melhores resultados do estado arte, o VerdictDB [44], como método de comparação. Para a realização das experiências foi selecionado o *Benchmark* TPC-H [13], que possui a especificação de vinte e duas pesquisas de negócio, simulando um sistema de apoio à decisão que explora uma grande quantidade de dados. O principal objetivo a atingir com a realização destas experiências é testar os dois tipos de operações mais utilizados com grandes volumes de dados: funções de agregação e junções. Estas são as operações que têm maior tempo de processamento , devido aos custos consideráveis de processamento e de Entrada/Saída. As pesquisas utilizadas recorrem a diferentes funções de agregação e a diferentes tamanhos de junções para o mesmo conjunto de dados, sendo úteis para avaliar o *speedup* de modo a identificar qual a diferença entre o ganho do VerdictDB e da nova abordagem. Além disso, as pesquisas utilizadas também permitem calcular o erro relativo percentual das pesquisas, para cada uma das abordagens.
Durante a avaliação experimental o grau de confiança e o erro máximo admitido não foram considerados, pois o VerdictDB não possui esta configuração, não sendo correto efetuar uma comparação utilizando estes dois parâmetros. Assim, as amostras do JDBCApprox foram geradas utilizando sempre 20% dos dados da tabela que possui um maior número de chaves estrangeiras. As pesquisas Q17, Q20 e Q21, apresentadas no Anexo B, não foram consideradas na avaliação experimental devido ao seu elevado tempo de execução. Após oito horas não foi possível obter o resultado para essas pesquisas, quando executadas no PostgreSQL, sendo a sua execução abortada.

5.2 Ambiente experimental

Esta secção apresenta as caraterísticas do ambiente experimental. Inicialmente é apresentada uma descrição do *benchmark* utilizado, juntamente com o esquema da base de dados. Em seguida são descritas as caraterísticas das máquinas utilizadas no cenário experimental. Posteriormente são enumeradas as duas métricas utilizadas: *speedup* e erro relativo percentual e é descrita a metodologia seguida durante a avaliação experimental.

5.2.1 *Benchmark* TPC-H

Para realizar as experiências foi necessário selecionar uma base de dados com o tamanho que justificasse a utilização de um processamento aproximado de pesquisas. Assim, o *benchmark* TPC-H [13] foi selecionado para a avaliação experimental pelo seu gerador de dados, que permitiu utilizar o mesmo conjunto de dados para diferentes quantidades de dados e pelas pesquisas com um alto grau de complexidade.
O *benchmark* TPC-H apresenta um conjunto de pesquisas de avaliam o desempenho dos sistemas de apoio à decisão, tendo sido proposto pelo *Transaction Processing Performance Council* em novembro de 1999. Estas pesquisas produzem respostas às questões mais críticas do negócio.

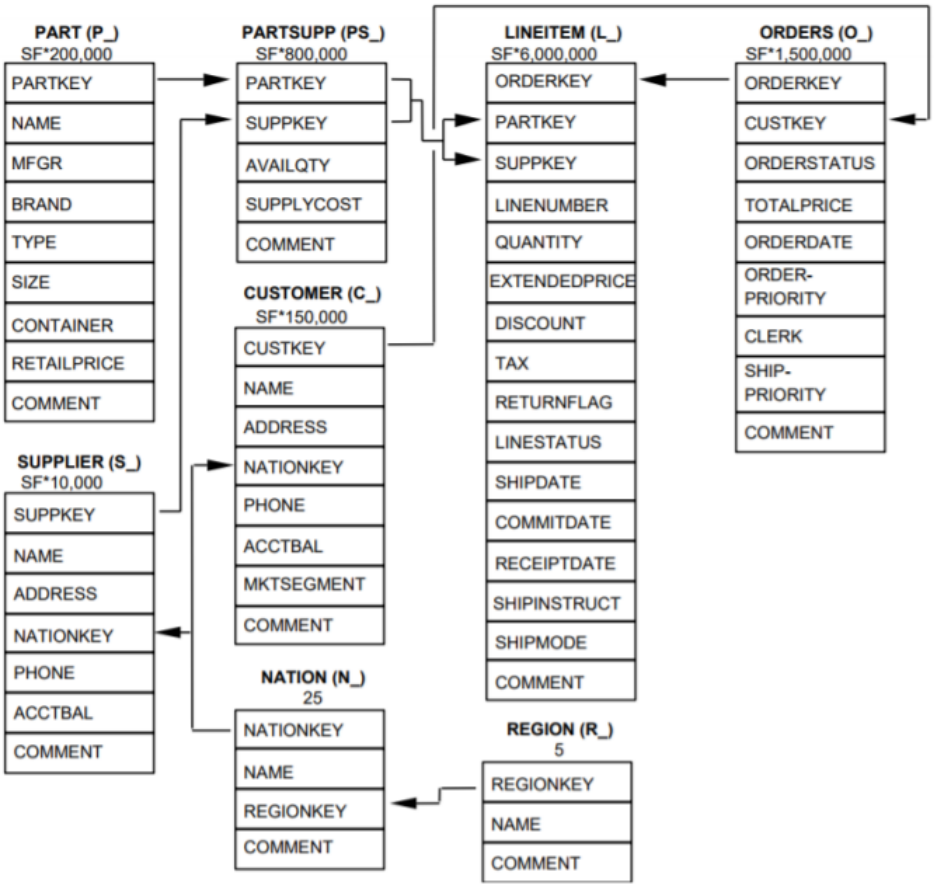

Figura 5.1: Esquema da base de dados do *benchmark* TPC-H

A Figura 5.1 representa o esquema da base de dados utilizado. Para além da especificação dos nomes e atributos de cada tabela, esta figura fornece informações valiosas sobre o tamanho da base de dados. É visível em cima de cada tabela a fórmula para obter o número de registos presentes em cada tabela, de acordo com fator de escala usado. Por exemplo, as tabelas *Nation* e *Region* têm tamanhos fixos de 5 e 25 respetivamente, enquanto que a

tabela *Part* possui 200000 registos para um fator de escala igual a 1 (1 GB de dados). Na avaliação experimental o *benchmark* TPC-H foi implementado com dois fatores de escala: 1 e 10, correspondendo respetivamente a 1 e 10GB para o tamanho da base de dados. As 22 pesquisas utilizadas nesta avaliação estão apresentadas no Anexo A. Estas pesquisas possuem caraterísticas variadas, consultando uma ou várias tabelas com um grande número de restrições presentes.

5.2.2 Caraterísticas do ambiente experimental

As tabelas 5.1 e 5.2 especificam os principais recursos das máquinas 1 e 2, respetivamente. Estas máquinas foram utilizadas com o objetivo de separar a base de dados original que estava presente no PostgreSQL das amostras que seriam copiadas para o SQLite, garantindo assim mais memória disponível na máquina 2.

Tabela 5.1: Recursos da máquina 1

Recursos da Máquina 1	
CPU	2.19 GHz Intel Xeon E312xx (Sandy Bridge)
RAM	4GB
Sistema Operativo	Ubuntu 16.04 64-bit
PostgreSQL	Versão 10.9

Tabela 5.2: Recursos da máquina 2

Recursos da Máquina 2	
CPU	2.19 GHz Intel Xeon E312xx (Sandy Bridge)
RAM	8GB
Sistema Operativo	Ubuntu 16.04 64-bit
Java	Versão 8
SQLite	Versão 3
VerdictDB	Versão 0.5.8

5.2.3 Métricas utilizadas

Na avaliação experimental foram utilizadas duas métricas: *speedup* e erro relativo percentual. Em seguida é descrita cada uma delas.

Speedup

O *speedup* mede o desempenho relativo de dois sistemas que processam o mesmo problema, ou seja, é a melhoria na velocidade de execução de uma tarefa executada em duas arquiteturas similares com recursos semelhantes [10]. Na avaliação experimental foram calculados o *speedup* do VerdictDB, através da equação 5.1 e do JDBCApprox relativamente ao PostgreSQL, através da equação 5.2.

$$Speedup\ do\ VerdictDB = \frac{Tempo\ de\ resposta\ do\ PostgreSQL}{Tempo\ de\ resposta\ do\ VerdictDB} \tag{5.1}$$

$$Speedup\ do\ JDBCApprox = \frac{Tempo\ de\ resposta\ do\ PostgreSQL}{Tempo\ de\ resposta\ do\ JDBCApprox} \tag{5.2}$$

Os valores de *speedup* permitiram comparar a diferença entre o ganho do VerdictDB e do JDBCApprox. Um valor de *speedup* inferior a um significa que não existiu qualquer ganho,

ou seja, a abordagem utilizada possui um tempo de resposta superior ao do PostgreSQL. Em contrapartida, um valor de *speedup* superior a um significa que a abordagem utilizada possui um tempo de resposta inferior ao do PostgreSQL.
O tempo de criação das amostras não foi considerado nos valores de *speedup* do JDBCApprox e do VerdictDB.

Erro relativo percentual

O erro relativo percentual permite identificar qual a percentagem de erro de um valor aproximado, relativamente a um valor exato. Na avaliação experimental foram calculados o erro relativo percentual da resposta aproximada às pesquisas devolvida pelo VerdictDB e pelo JDBCApprox, em relação à resposta exata devolvida pelo PostgreSQL. O erro relativo percentual de uma pesquisa permitiu avaliar a exatidão das respostas das duas abordagens analisadas e foi calculado através da equação 5.3.

$$Erro\ relativo\ percentual = \left| \frac{Resposta\ Exata - Resposta\ Aproximada}{Resposta\ Exata} \right| \times 100 \quad (5.3)$$

5.2.4 Metodologia

Na avaliação experimental foram utilizados três cenários distintos:

- **Cenário 1: PostgreSQL**
  Este cenário recorre a uma aplicação Java (máquina 1) para efetuar as pesquisas na base de dados presente no PostgreSQL (máquina 1);
- **Cenário 2: VerdictDB**
  Este cenário recorre a uma aplicação Java (máquina 1) para criar amostras na base de dados presente no PostgreSQL (máquina 1) através do VerdictDB. Em seguida as pesquisas são realizadas a essas amostras, recorrendo novamente ao VerdictDB;
- **Cenário 3: JDBCApprox**
  Este cenário recorre a uma aplicação Java (máquina 1) para criar amostras na base de dados presente no PostgreSQL (máquina 1) através do JDBCApprox, que em seguida são copiadas para o SQLite com uma configuração em memória (máquina 2). Posteriormente as pesquisas são realizadas recorrendo às amostras presentes no SQLite.

As pesquisas presentes no Anexo B foram executadas trinta vezes em cada cenário experimental. Para a execução de uma nova pesquisa o serviço do PostgreSQL era reiniciado, de modo a que todas as pesquisas fossem executadas nas mesmas condições. Nos cenários 1 e 2 foi excluída a primeira execução, uma vez que o PostgreSQL carrega os dados do disco para memória, sendo este um processo mais lento. Para cada pesquisa, após as trinta execuções, foi calculada a média do resultado e do tempo de resposta.
As pesquisas Q17, Q20 e Q21 não foram utilizadas na avaliação experimental, uma vez que não eram executadas no PostgreSQL e não era possível obter o valor exato da resposta.
No cenário 2 não foram utilizadas as pesquisas Q7, Q8, Q16 e Q22 uma vez que o VerdictDB não executa pesquisas que referenciam a mesma tabela mais do que uma vez.

5.3 Discussão de resultados

Esta secção apresenta os resultados obtidos durante a avaliação experimental. Inicialmente é analisado o tempo de criação de amostras de cada tabela no JDBCApprox e no VerdictDB para um total de 1GB e 10GB de dados. Em seguida é realizada uma análise do erro relativo percentual no resultado de cada pesquisa. No final é avaliado o *speedup* para cada pesquisa e são apresentadas as conclusões da avaliação experimental.

5.3.1 Criação de amostras

A criação de uma amostra no JDBCApprox consiste em criar uma tabela com a mesma estrutura da tabela original, mas com uma menor quantidade de dados. Em contrapartida, a criação de uma amostra no VerdictDB consiste em criar uma tabela denominada *scramble*, que possui a mesma estrutura e dados da tabela original, no entanto a ordem dos registos da tabela é trocada, de modo a que os dados estejam dispostos numa ordem diferente da tabela original. Além disso, a tabela *scramble* possui uma coluna adicional, que indica a que bloco pertence cada linha da tabela, de modo a que os dados se encontrem divididos por blocos. Os dados são executados bloco a bloco e é calculada uma estimativa de erro, quando o erro for inferior a 5%, ou todos os dados forem processados, o VerdictDB retorna um resultado.
O gráfico presente na figura 5.2 ilustra o tempo de criação das amostras de cada tabela, para o JDBCApprox e para o VerdictDB com 1GB de dados. É possível visualizar uma enorme discrepância do tempo de criação da amostra da tabela *Lineitem* utilizando o VerdictDB em relação aos restantes tempos de criação. Esta diferença relaciona-se com o facto da tabela *Lineitem* ser a maior na base de dados (possui 6.001.215 linhas) e a tabela de amostra no VerdictDB possui o mesmo número de linhas. No JDBCApprox criar uma amostra da tabela *Lineitem* é cerca de seis vezes mais rápido do que no VerdictDB, uma vez que a amostra irá possuir 20% dos dados originais (1.200.243 linhas).
Os tempo de criação de amostras com o VerdictDB encontra-se de acordo com o esperado, ou seja, a criação de amostras das tabelas que possuem um maior número de registos é mais lenta do que as restantes. Em contrapartida, o tempo de criação de amostras com o JDBCApprox não tem o mesmo comportamento. Ou seja, a maior tabela da base de dados é criada através da seleção de modo aleatório de 20% dos dados da tabela original, sendo este um processo mais rápido. As restantes tabelas são criadas a partir das junções com outra tabela, demorando mais tempo.
A criação de amostras de todas as tabelas presentes na base de dados para 1GB de dados demora cerca de 39.125ms com o VerdictDB e 23.611ms com o JDBCAproox.

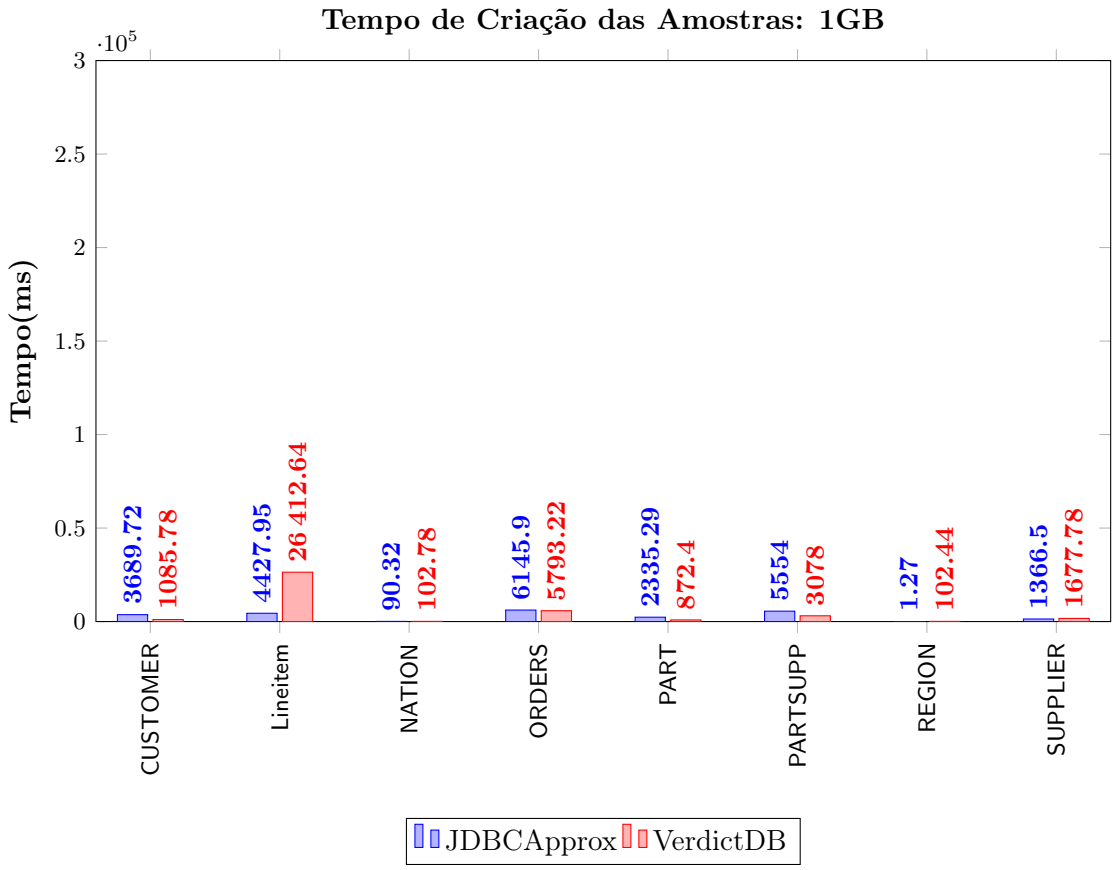

Figura 5.2: Tempo de criação das amostras de cada tabela para 1GB de dados

O gráfico presente na figura 5.3 ilustra o tempo de criação das amostras de cada tabela, para o JDBCApprox e para o VerdictDB com 10GB de dados. Os tempos de criação têm o mesmo comportamento do gráfico anterior, com tempos de criação mais elevados, uma vez que a quantidade dos dados é dez vezes superior.
A criação de amostras de todas as tabelas presentes na base de dados para 10GB de dados demora cerca de 365.949ms com o VerdictDB e 399.167ms com o JDBCApprox.

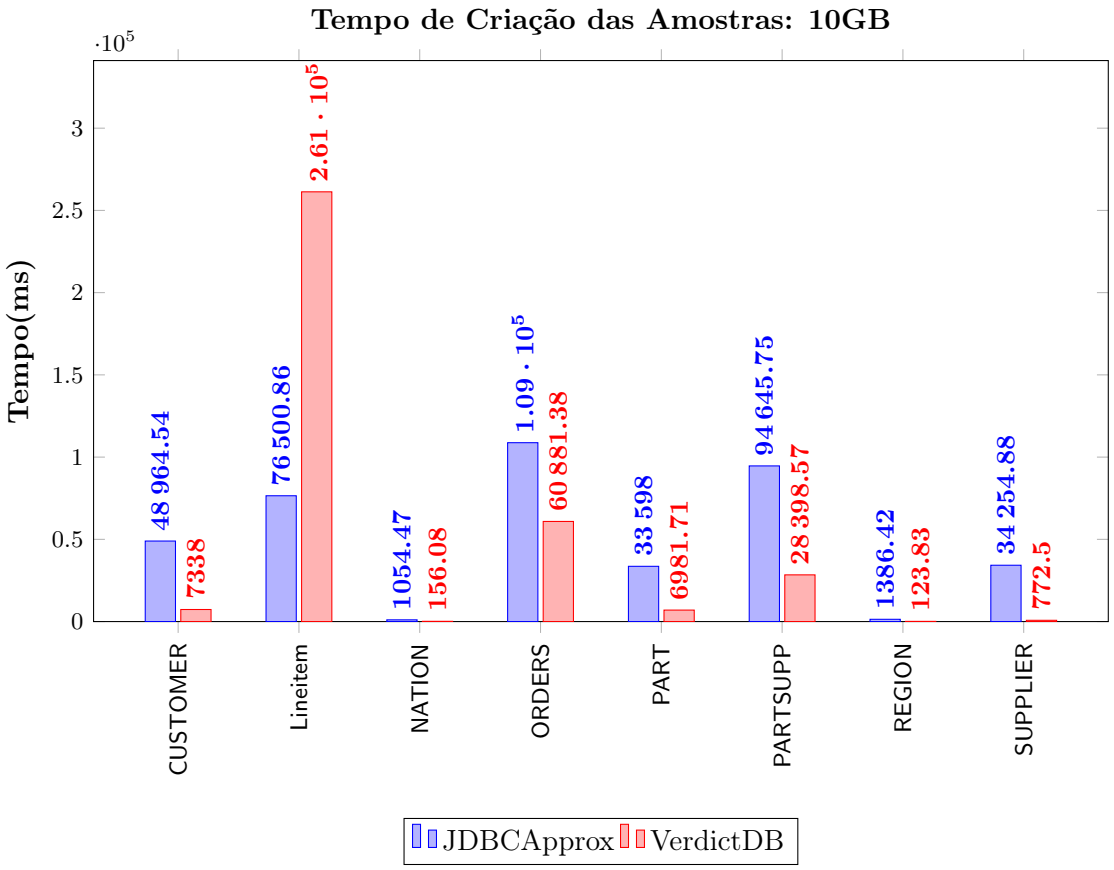

Figura 5.3: Tempo de criação das amostras de cada tabela para 10GB de dados

O tempo de criação total das amostras com o JDBCApprox e com o VerdictDB são semelhantes, no entanto o JDBCApprox é mais rápido. A criação de amostras de todas as tabelas presentes na base de dados para 10GB de dados demora cerca de 365.948ms com o VerdictDB e 399.166ms com o JDBCApprox.

5.3.2 Erro no resultado das pesquisas

Em seguida são apresentados os resultados obtidos relativamente ao erro relativo percentual no resultado de cada pesquisa.

Pesquisa Q1

A tabela 5.3 apresenta o erro relativo percentual registado na pesquisa Q1 utilizando o JDBCApprox, para cada grupo e para as funções de agregação: SUM, COUNT e AVG. Este erro é muito reduzido quando são utilizados diferentes tamanhos de dados, uma vez que a pesquisa Q1 recorre apenas à tabela *Lineitem* e utiliza uma restrição na cláusula *where*. Estas caraterísticas da pesquisa permitem que sejam abrangidas mais quantidades de dados dos 20% presentes na amostra para responderem à pesquisa.
A função de agregação AVG é a que apresenta valores menores para o erro relativo percentual (entre 0.01% e 0.06%), uma vez que estatisticamente, no cálculo de um valor médio existe uma menor dispersão dos valores.
Os valores do erro relativo percentual mais elevados para a função SUM estão relacionados com a quantidade de dados que são incluídos em cada grupo. Os grupos 2 (onde, *l_returnflag*='N' e *l_linestatus*='F') e 3 (onde, *l_returnflag*='N' e *l_linestatus*='O') são os que abrangem menos registos da base de dados, resultando por isso num valor estimado

com um erro superior.

Tabela 5.3: Erro relativo percentual na pesquisa Q1 para o JDBCApprox

	Grupo	**SUM(%)**	**COUNT(%)**	**AVG(%)**
1GB	1	0.01	0.03	0.04
	2	0.49	0.13	0.06
	3	0.10	0.04	0.05
	4	0.03	0.05	0.06
10GB	1	0.02	0.01	0.03
	2	0.11	0.01	0.06
	3	0.45	0.02	0.02
	4	0.02	0.02	0.01

A tabela 5.4 apresenta o erro relativo percentual registado na pesquisa Q1 utilizando o VerdictDB, para cada grupo e para as funções de agregação: SUM, COUNT e AVG.
Os valores do erro são novamente mais elevados nos grupos que abrangem uma menor quantidade de registos da tabela, resultando em respostas estimadas de forma menos exata.

Tabela 5.4: Erro relativo percentual na pesquisa Q1 para o VerdictDB

	Grupo	**SUM(%)**	**COUNT(%)**	**AVG(%)**
1GB	1	0.02	0.04	0.01
	2	0.63	0.28	0.05
	3	0.37	0.06	0.11
	4	0.02	0.03	0.01
10GB	1	0.09	0.03	0.06
	2	0.32	0.17	0.16
	3	0.39	0.04	0.08
	4	0.02	0.08	0.01

O erro relativo percentual nas duas abordagens comparadas é muito semelhante, não existindo nenhuma diferença superior a 0.16%. No entanto, esta diferença poderá estar relacionada com os dados que foram selecionados para responder à amostra, que em ambas as abordagens são elegidos de forma aleatória.

Pesquisa Q2

Os gráficos apresentados nas figuras 5.4 e 5.5 apresentam o erro relativo percentual registado na pesquisa Q2 utilizando o VerdictDB e o JDBCApprox para 1GB e 10GB, respetivamente. O erro é apresentado para cada grupo (*France*, *Germany*, *Romania*, *Russia* e *UK*) e para a função de agregação SUM.

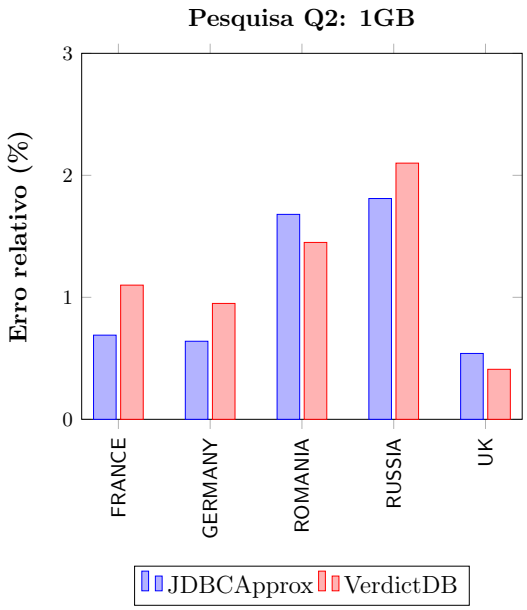

Figura 5.4: Erro relativo percentual na pesquisa Q2 com 1GB de dados

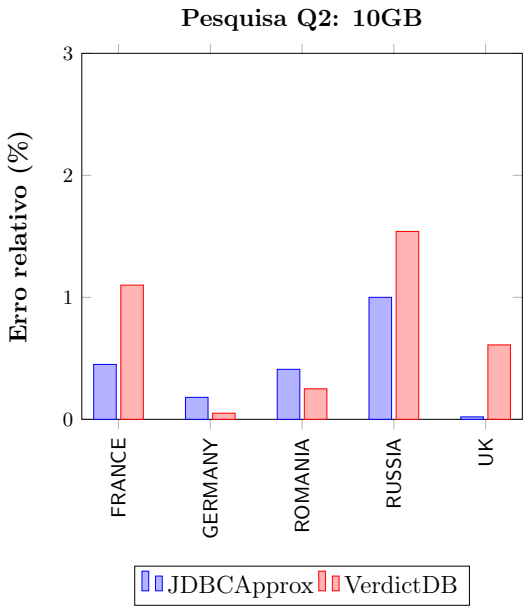

Figura 5.5: Erro relativo percentual na pesquisa Q2 com 10GB de dados

De um modo geral, o erro relativo percentual diminui à medida que aumenta a quantidade de dados, uma vez que a amostra selecionada irá possuir mais registos da base de dados. Quando a pesquisa Q2 é efetuada no VerdictDB e posteriormente no JDBCApprox, o erro relativo percentual registado é na maior parte dos casos superior no VerdictDB. Os diferentes valores de erro estão relacionados com o modo como o VerdictDB utiliza os dados para responder à pesquisa, uma vez que divide os dados por bloco e executa-os, bloco a bloco até os valores convergirem e o erro ser inferior a um valor definido.
O erro relativo percentual para o grupo *RUSSIA* é o mais elevado, pois este é o grupo com menor número de elementos da pesquisa.

Pesquisa Q3

Os gráficos apresentados nas figuras 5.6 e 5.7 apresentam o erro relativo percentual registado na pesquisa Q3 utilizando o VerdictDB e o PostgreSQL para 1GB e 10GB, respetivamente. O erro é apresentado para cada grupo (*AUTOMOBILE*, *BUILDING*).

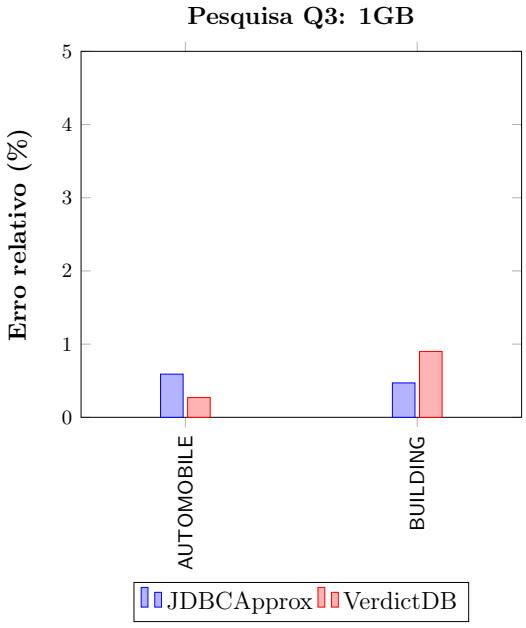

Figura 5.6: Erro relativo percentual na pesquisa Q3 com 1GB de dados

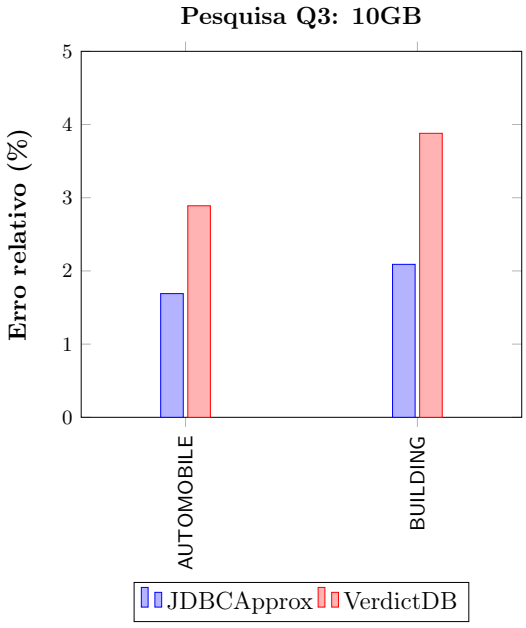

Figura 5.7: Erro relativo percentual na pesquisa Q3 com 10GB de dados

A pesquisa Q3 possui praticamente o mesmo número de registos em cada grupo, pelo que o erro relativo não é influenciado por esse fator. Este valor de erro é mais elevado quando se utilizam 10GB de dados, uma vez que os valores selecionados para a amostra poderiam não ter sido os que representariam melhor a tabela original. Além disso, é estimada a soma de dois valores, sendo o erro relativo mais elevado.

Pesquisa Q4

Os gráficos apresentados nas figuras 5.8 e 5.9 apresentam o erro relativo percentual registado na pesquisa Q4 utilizando o VerdictDB e o JDBCApprox para 1GB e 10GB, respetivamente. O erro é apresentado para cada grupo (*1-URGENT*, *2-HIGH*, *3-MEDIUM*, *4-NOT SPECIFIED*, *5-LOW*).

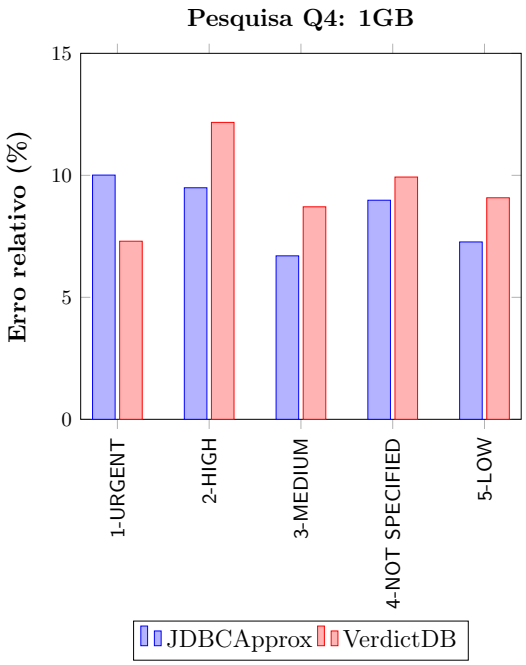

Figura 5.8: Erro relativo percentual na pesquisa Q4 com 1GB de dados

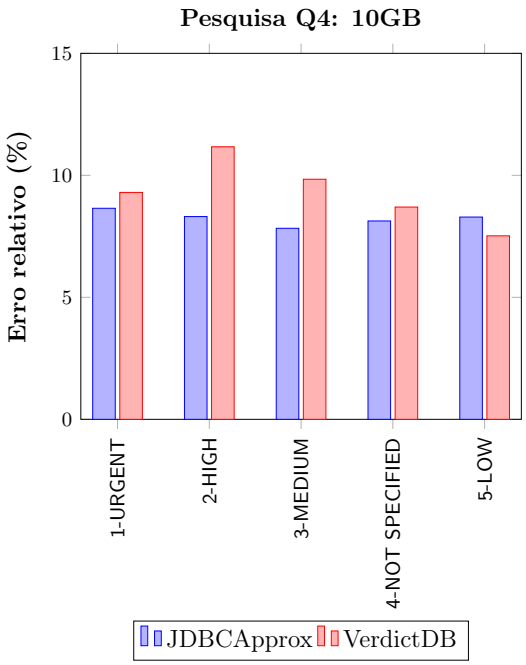

Figura 5.9: Erro relativo percentual na pesquisa Q4 com 10GB de dados

A pesquisa Q4 recorre a uma sub-pesquisa que seleciona todos os registos da tabela *Lineitem* de acordo com duas restrições, que posteriormente vão influenciar o erro relativo. Ou seja, como a tabela *Lineitem* é uma amostra da tabela original, não vai possuir todos os registos necessários para a pesquisa principal. Além disso, esta pesquisa abrange poucos registos devido à restrição das datas, que se referem a três meses.

Pesquisa Q5

Os gráficos apresentados nas figuras 5.10 e 5.11 apresentam o erro relativo percentual registado na pesquisa Q5 utilizando o VerdictDB e o JDBCApprox para 1GB e 10GB, respetivamente. O erro é apresentado para cada grupo (*China*, *India*, *Indonesia*, *Japan*, *Vietnam*).

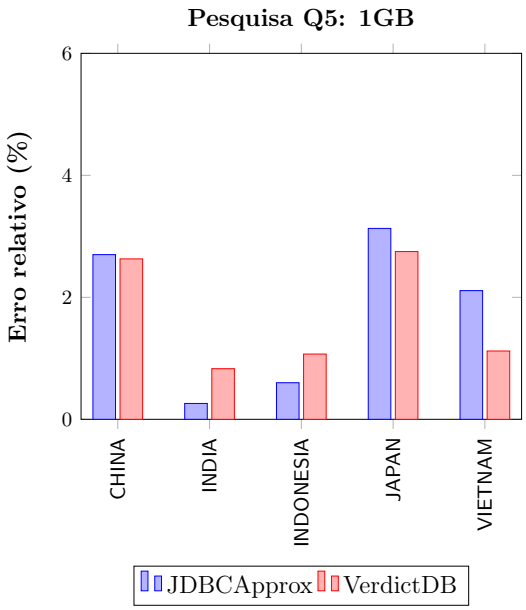

Figura 5.10: Erro relativo percentual na pesquisa Q5 com 1GB de dados

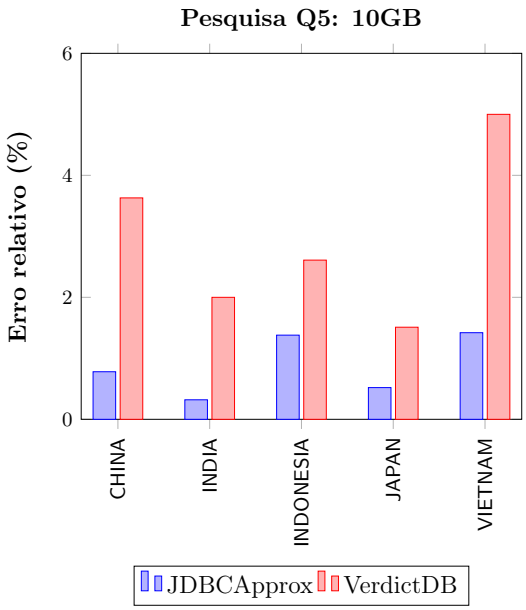

Figura 5.11: Erro relativo percentual na pesquisa Q5 com 10GB de dados

A pesquisa Q5 utiliza seis tabelas e muitas restrições na cláusula *where*, que influenciam o erro relativo. Este atinge um valor máximo de 3% para o JDBCApprox e de 5% para o VerdictDB. O grupo *Japan* possui um menor número de registos, registando um erro relativo mais elevado com 1GB de dados.

Pesquisa Q6

Os gráficos apresentados nas figuras 5.12 e 5.13 apresentam o erro relativo percentual registado na pesquisa Q6 utilizando o VerdictDB e o JDBCApprox para 1GB e 10GB, respetivamente.

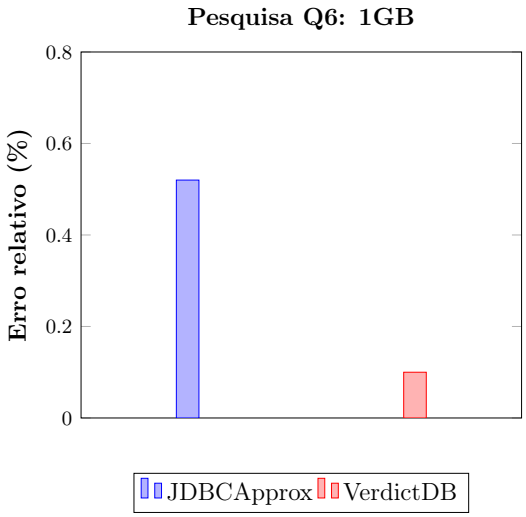

Figura 5.12: Erro relativo percentual na pesquisa Q6 com 1GB de dados

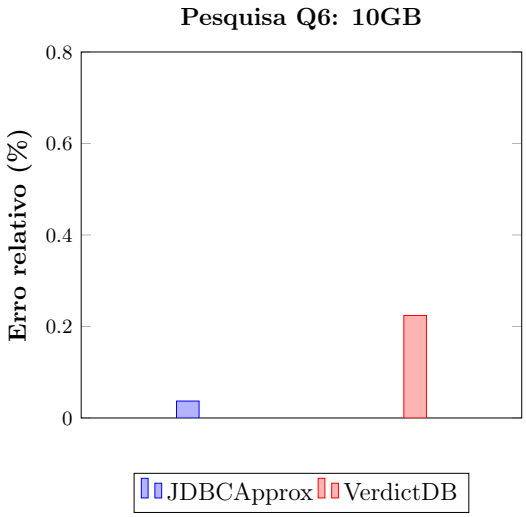

Figura 5.13: Erro relativo percentual na pesquisa Q6 com 10GB de dados

Apesar do erro relativo percentual utilizando o VerdictDB ser diferente daquele que é obtido

quando é utilizado o JDBCApprox, este valor não é significativo. Quando a pesquisa Q6 é executada com apenas 1GB de dados existe uma diferença de cerca de 0.4%, em contrapartida essa diferença diminui para cerca de 0.15% quando são utilizados 10GB de dados. Esta diferença está relacionada com a aleatoriedade dos dados selecionados para responder à pesquisa.

Pesquisa Q7

A pesquisa Q7 não é executada no VerdictDB, uma vez que a tabela *Nation* é referenciada duas vezes na sub-pesquisa. Os gráficos apresentados nas figuras 5.14 e 5.15 apresentam o erro relativo percentual registado na pesquisa Q7 utilizando o JDBCApprox para 1GB e 10GB, respetivamente. O erro é apresentado para cada grupo (Grupo 1, Grupo 2, Grupo 3, Grupo 4).

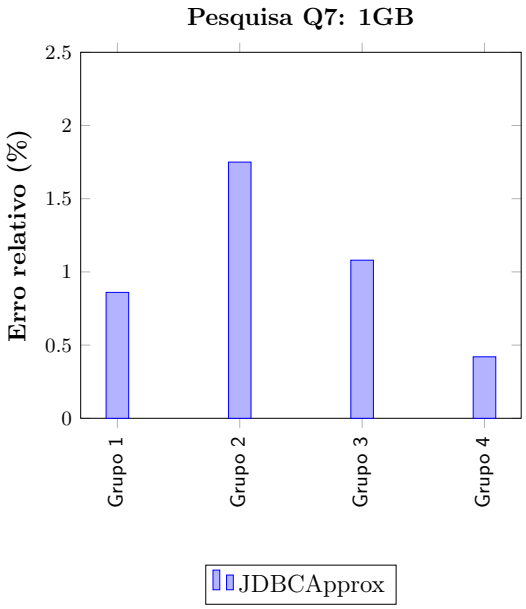

Figura 5.14: Erro relativo percentual na pesquisa Q7 com 1GB de dados

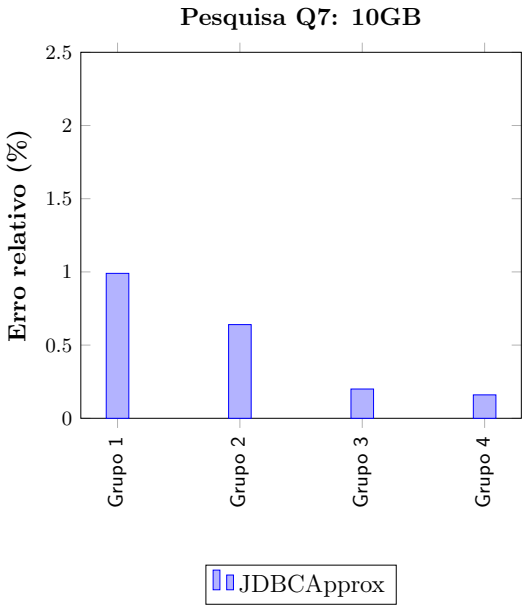

Figura 5.15: Erro relativo percentual na pesquisa Q7 com 10GB de dados

O erro relativo percentual obtido para a pesquisa Q7 é maior do que o erro correspondente, por exemplo à pesquisa Q3, uma vez que o número médio de registos agregados por grupo na pesquisa Q7 é menor.

Pesquisa Q8

A pesquisa Q7 não é executada no VerdictDB, uma vez que a tabela *Nation* é referenciada duas vezes na sub-pesquisa. Os gráficos apresentados nas figuras 5.16 e 5.17 apresentam o erro relativo percentual registado na pesquisa Q7 utilizando o JDBCApprox para 1GB e 10GB, respetivamente. O erro é apresentado para cada grupo (1995, 1996).

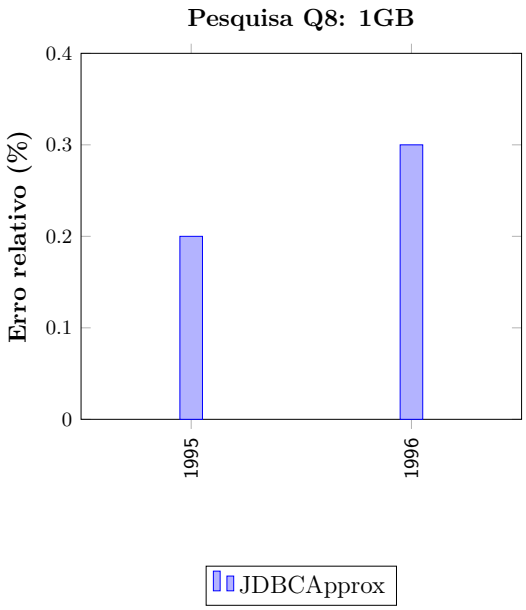

Figura 5.16: Erro relativo percentual na pesquisa Q8 com 1GB de dados

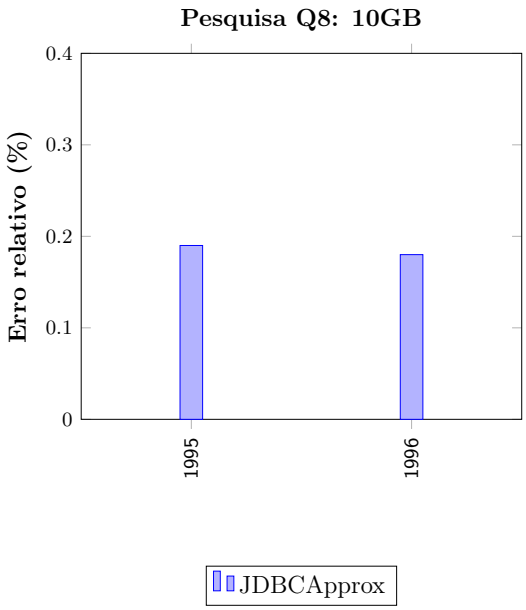

Figura 5.17: Erro relativo percentual na pesquisa Q8 com 10GB de dados

O erro relativo percentual obtido para a pesquisa Q8 é considerado bastante bom, uma vez que não ultrapassa os 0.3%. Este valor está relacionado com o facto de esta pesquisa efetuar uma soma quando existem registos pertencentes ao Brasil da tabela *nation*, sendo que estes são os que existem em maior quantidade.

Pesquisa Q9

Os gráficos apresentados nas figuras 5.18 e 5.19 apresentam o erro relativo percentual registado na pesquisa Q9 utilizando o VerdictDB e o JDBCApprox para 1GB e 10GB, respetivamente. O erro é apresentado para cada grupo (Grupo 1, Grupo 2, Grupo 3, Grupo 4, Grupo 5).

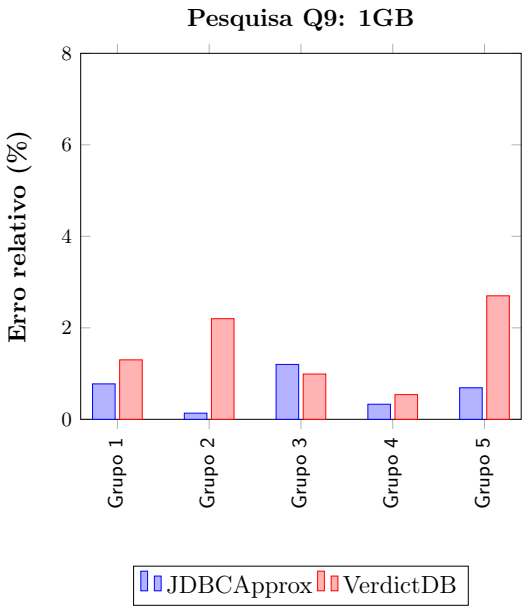

Figura 5.18: Erro relativo percentual na pesquisa Q9 com 1GB de dados

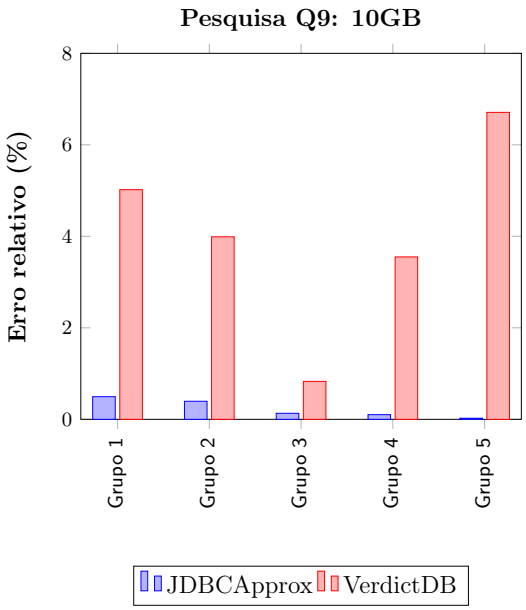

Figura 5.19: Erro relativo percentual na pesquisa Q9 com 10GB de dados

A pesquisa Q9 utiliza uma sub-pesquisa que consulta seis tabelas. O VerdictDB apresenta um erro relativo maior do que o JDBCApprox, uma vez que a execução dos dados bloco a bloco não é muito eficaz quando se utilizam sub-pesquisas. Os blocos utilizados na sub-pesquisa podem não coincidir com os que são utilizados na pesquisa, resultando em respostas menos exatas.

Pesquisa Q10

A pesquisa Q10 agrupa os resultados de acordo com os atributos *c_custkey*, *c_name* e *revenue*. Os dois primeiros atributos são únicos, ou seja, não existem valores de *c_custkey* e *c_name* iguais. Deste modo, como se utiliza uma amostra dos dados selecionada de forma aleatória, não é garantido que o JDBCApprox consiga incluir nessa amostra exatamente os mesmos dados que estão presentes na figura 5.20. No caso do VerdictDB, como todos os dados estão presentes na tabela da amostra, o resultado apresentado é igual ao resultado original.

c_custkey	c_name	revenue	c_acctbal
61453	Customer#000061453	97771.88	9999.99
144232	Customer#000144232	133066.06	9999.74
129934	Customer#000129934	56209.86	9999.59
15980	Customer#000015980	86897.50	9999.23
96205	Customer#000096205	122898.60	9998.86

Figura 5.20: Resultado exato da pesquisa Q10

Para esta pesquisa não foi calculado o erro relativo percentual, uma vez que os grupos selecionados eram em grande parte distintos dos grupos da resposta original. No entanto, durante as trinta execuções o VerdictDB devolveu todas as respostas com os cinco registos iguais ao resultado original. Em contrapartida, o JDBCApprox devolveu a maioria das

respostas com três registos iguais ao resultado original.

Pesquisa Q11

Na pesquisa Q11, à semelhança da pesquisa anterior, o erro relativo percentual não foi calculado. Nesta pesquisa os valores para o atributo *ps_partkey* devolvidos pelas respostas aproximadas eram distintos daqueles que a resposta original devolvia. A resposta exata está presente na figura 5.21.

ps_partkey	value
129760	17538456.86
166726	16503353.92
191287	16474801.97
161758	16101755.54
34452	15983844.72

Figura 5.21: Resultado exato da pesquisa Q11

Durante as trinta execuções o JDBCApprox devolveu três registos iguais ao do resultado original em cada resposta. O VerdictDB não suporta a pesquisa Q11, uma vez que esta utiliza uma sub-pesquisa que referencia a mesma tabela da pesquisa.

Pesquisa Q12

Os gráficos apresentados nas figuras 5.22 e 5.23 apresentam o erro relativo percentual registado na pesquisa Q12 utilizando o VerdictDB e o JDBCApprox para 1GB e 10GB, respetivamente. O erro é apresentado para cada grupo (Grupo 1 - *High*, Grupo 1 - *Low*, Grupo 2 - *High*, Grupo 2 - *Low*).

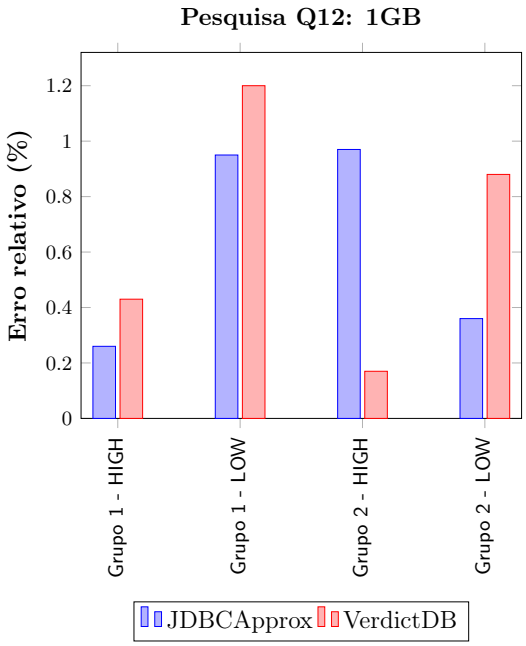

Figura 5.22: Erro relativo percentual na pesquisa Q12 com 1GB de dados

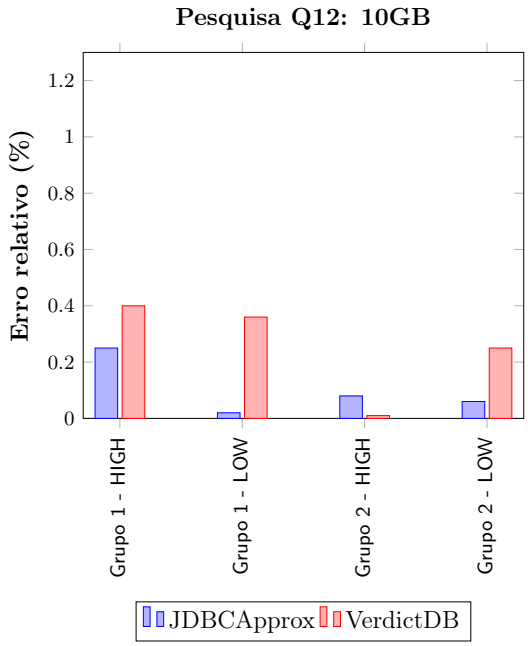

Figura 5.23: Erro relativo percentual na pesquisa Q12 com 10GB de dados

A pesquisa Q12 utiliza apenas duas tabelas para obter uma resposta. Esta consiste em somar os valores 0 ou 1 de acordo com determinadas restrições. Como estes valores são apenas dois e não variam, o erro relativo percentual é reduzido.

Pesquisa Q13

A pesquisa Q13, para além de recorrer à função de agregação COUNT, seleciona o atributo *c_ count*, como mostra a figura 5.24. Quando esta é executada com o JDBCApprox existe novamente uma falta de atributos importantes para obter a sua resposta, neste caso o atributo *c_ count*.

c_count	custdist
0	50005
9	6641
10	6532
11	6014
8	5937

Figura 5.24: Resultado exato da pesquisa Q13

Em cada execução com o JDBCApprox são selecionados em média dois registos iguais aos do resultado original, num total de cinco registos. No VerdictDB são selecionados todos os registos que são iguais aos do resultado original.

Pesquisa Q14

Os gráficos apresentados nas figuras 5.25 e 5.26 apresentam o erro relativo percentual registado na pesquisa Q14 utilizando o VerdictDB e o JDBCApprox para 1GB e 10GB, respetivamente.

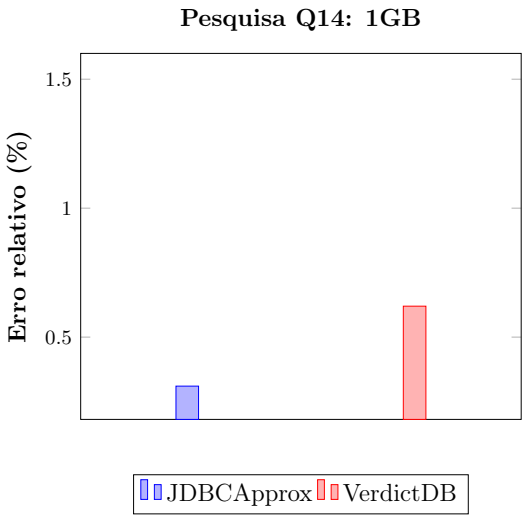

Figura 5.25: Erro relativo percentual na pesquisa Q14 com 1GB de dados

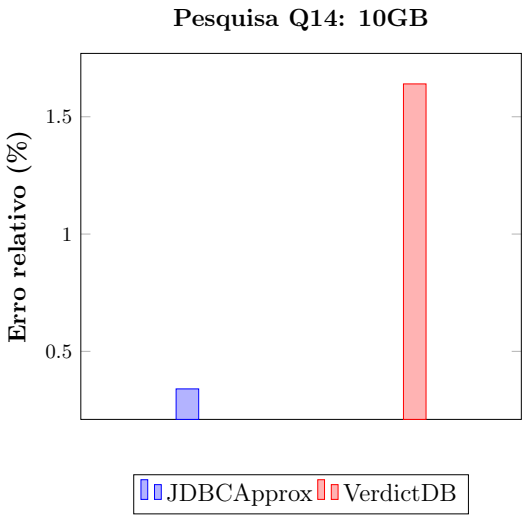

Figura 5.26: Erro relativo percentual na pesquisa Q14 com 10GB de dados

A pesquisa Q14 utiliza apenas duas tabelas e possui poucas restrições na cláusula *Where*, o que resulta em menores valores do erro relativo.

Pesquisa Q15

A pesquisa Q15 seleciona diretamente quatro atributos e calcula o atributo *max_value* selecionando o valor máximo de uma sub-pesquisa, sendo o seu resultado mostrado na figura 5.27. Esta pesquisa não é executada no VerdictDB, pois uma tabela é utilizada duas vezes.

s_suppkey	s_name	s_address	s_phone	max_value
8449	Supplier#000008449	Wp34zim9qYFbVctdW	20-469-856-8873	1772627.2087

Figura 5.27: Resultado exato da pesquisa Q15

Como o resultado da pesquisa é agrupado pelos quatro primeiros atributos, o VerdictDB nas trinta execuções devolve sempre um resultado exato, enquanto que o JDBCApprox apenas devolveu três vezes um resultado exato durante as trinta execuções.

Pesquisa Q16

Os gráficos apresentados nas figuras 5.28 e 5.29 apresentam o erro relativo percentual registado na pesquisa Q16 utilizando o JDBCApprox para 1GB e 10GB, respetivamente. O erro é apresentado para cada grupo (Grupo 1, Grupo 2, Grupo 3, Grupo 4, Grupo 5).

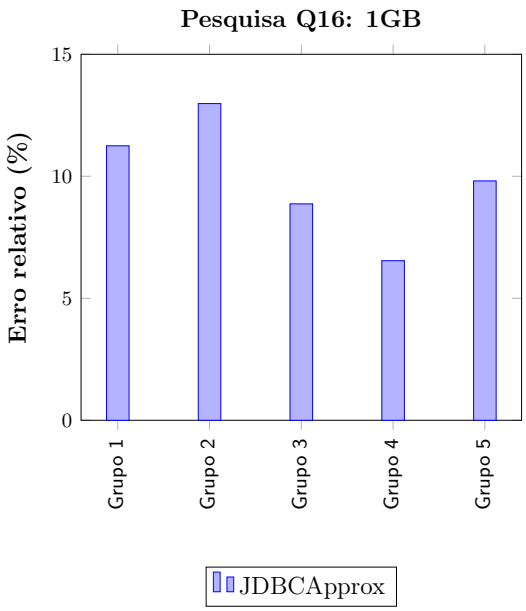

Figura 5.28: Erro relativo percentual na pesquisa Q16 com 1GB de dados

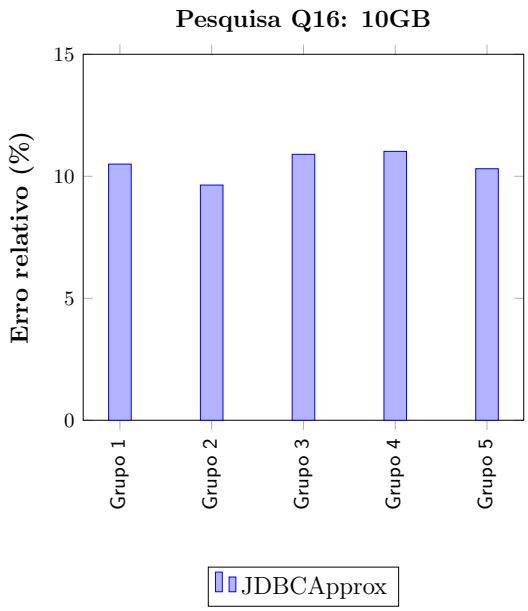

Figura 5.29: Erro relativo percentual na pesquisa Q16 com 10GB de dados

A pesquisa Q16 utiliza uma sub-pesquisa que seleciona as chaves primárias da tabela *Supplier*. Como foi utilizada uma amostra da tabela *Supplier* não é possível de garantir que essa amostra possua todos os valores de chave primária disponíveis. Isto origina um erro relativo elevado, sendo cerca de 11% quando são utilizados 10GB de dados.

Pesquisa Q18

A pesquisa Q18 não é suportada pelo VerdictDB, pois recorre à tabela *Lineitem* na pesquisa principal e na sub-pesquisa.

c_name	c_custkey	o_orderkey	o_orderdate	o_totalprice	sum
Customer#000128120	128120	4722021	1994-04-07	544089.09	323.00
Customer#000144617	144617	3043270	1997-02-12	530604.44	317.00
Customer#000013940	13940	2232932	1997-04-13	522720.61	304.00
Customer#000066790	66790	2199712	1996-09-30	515531.82	327.00
Customer#000046435	46435	4745607	1997-07-03	508047.99	309.00

Figura 5.30: Resultado exato da pesquisa Q18

Durante as trinta execuções, o JDBCApprox devolveu sempre três registos iguais aos apresentados na figura 5.30.

Pesquisa Q19

Os gráficos apresentados nas figuras 5.31 e 5.32 apresentam o erro relativo percentual registado na pesquisa Q19 utilizando o VerdictDB e o JDBCApprox para 1GB e 10GB, respetivamente.

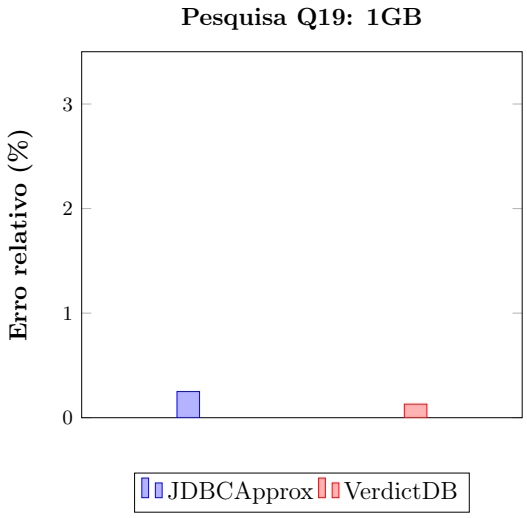

Figura 5.31: Erro relativo percentual na pesquisa Q19 com 1GB de dados

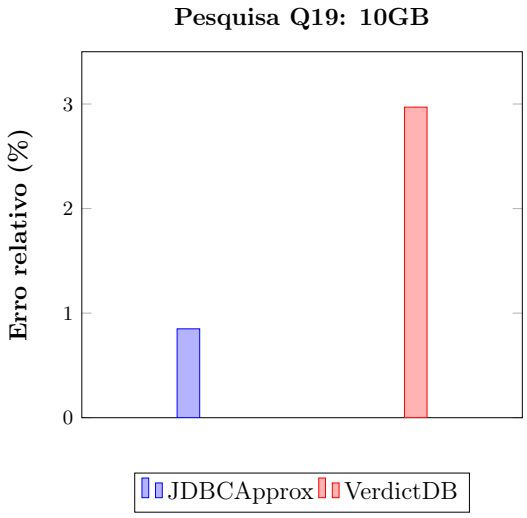

Figura 5.32: Erro relativo percentual na pesquisa Q19 com 10GB de dados

A pesquisa Q19 recorre a duas tabelas, sendo uma delas (*Lineitem* a que possui o maior número de registos da base de dados e a diversas restrições na cláusula *Where*. Para 1GB de dados o erro relativo percentual é muito reduzido para o JDBCApprox e para o VerdictDB, no entanto este cenário altera-se quando a quantidade de dados é aumentada para 10GB. Neste caso, o VerdictDB apresenta um erro relativo percentual de cerca de 3%. Este valor pode estar relacionado com a distribuição dos dados pelos blocos, uma vez que são utilizadas muitas restrições e os blocos podem ser utilizados com restrições diferentes.

Pesquisa Q22

Os gráficos apresentados nas figuras 5.33 e 5.34 apresentam o erro relativo percentual registado na pesquisa Q22 para a função de agregação COUNT, utilizando o JDBCApprox para 1GB e 10GB, respetivamente. O erro é apresentado para cada grupo (Grupo 1, Grupo 2, Grupo 3, Grupo 4, Grupo 5).

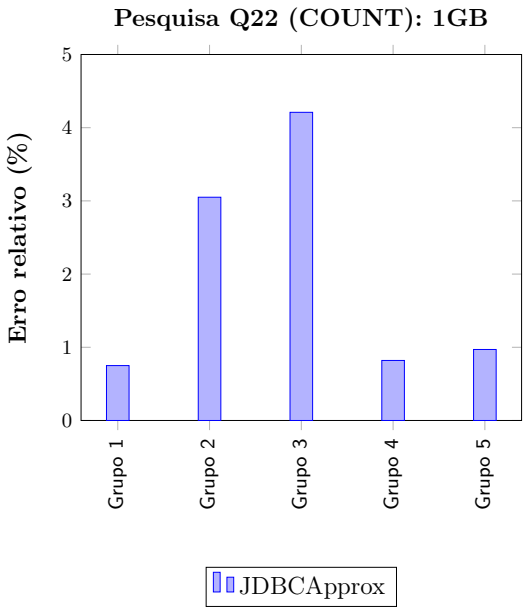

Figura 5.33: Erro relativo percentual na pesquisa Q22 para a função COUNT com 1GB de dados

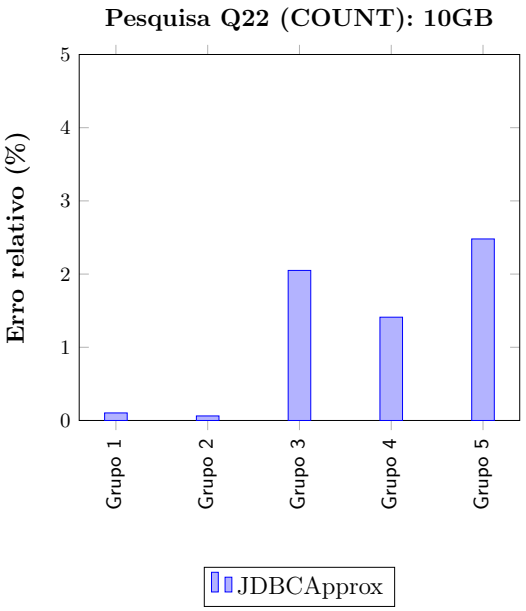

Figura 5.34: Erro relativo percentual na pesquisa Q22 para a função COUNT com 10GB de dados

Apesar da pesquisa Q22 ser complexa no sentido de utilizar duas sub-pesquisas, apenas recorre a duas tabelas no total e a poucas restrições na cláusula *Where*. Estas caraterísticas auxiliam a estimar uma resposta com um erro relativo menor.
Os gráficos apresentados nas figuras 5.35 e 5.36 apresentam o erro relativo percentual registado na pesquisa Q22 para a função de agregação SUM, utilizando o JDBCApprox para 1GB e 10GB, respetivamente. O erro é apresentado para cada grupo (Grupo 1, Grupo 2, Grupo 3, Grupo 4, Grupo 5).

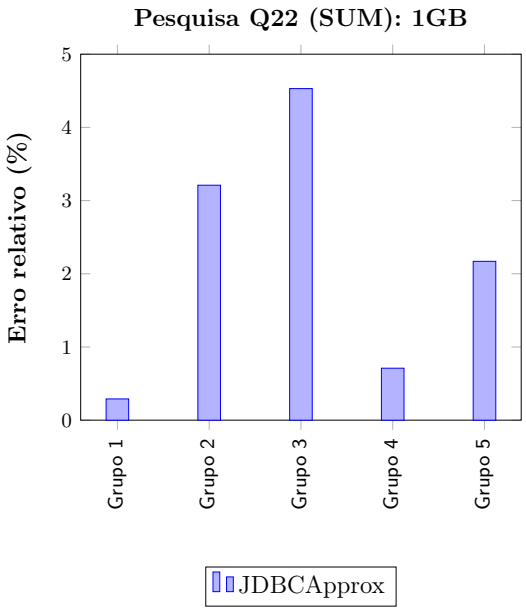

Figura 5.35: Erro relativo percentual na pesquisa Q22 para a função SUM com 1GB de dados

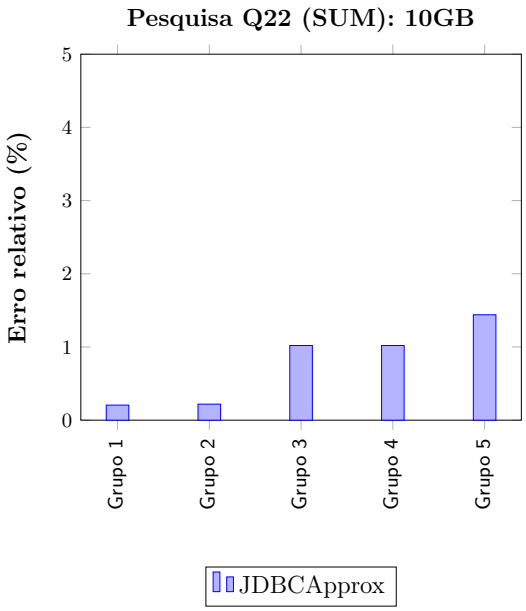

Figura 5.36: Erro relativo percentual na pesquisa Q22 para a função SUM com 10GB de dados

Para além das caraterísticas mencionadas anteriormente, o erro relativo de cada grupo é novamente influenciado pelo número de registos que consegue abranger. Com 1GB de dados existem mais registos para os grupos 1 e 4 e, com 10GB de dados existem mais registos para os grupos 1 e 2.

5.3.3 *Speedup*

Em seguida serão apresentados os resultados obtidos relativamente ao *speedup* na execução de cada pesquisa. As pesquisas Q7, Q8, Q11, Q16, Q18 e Q22 não são suportadas pelo VerdictDB, uma vez que utilizam uma sub-pesquisa que referencia a mesma tabela da pesquisa.
O gráfico da figura 5.37 ilustra o *speedup* para as pesquisas executadas no JDBCApprox e no VerdictDB para 1GB.

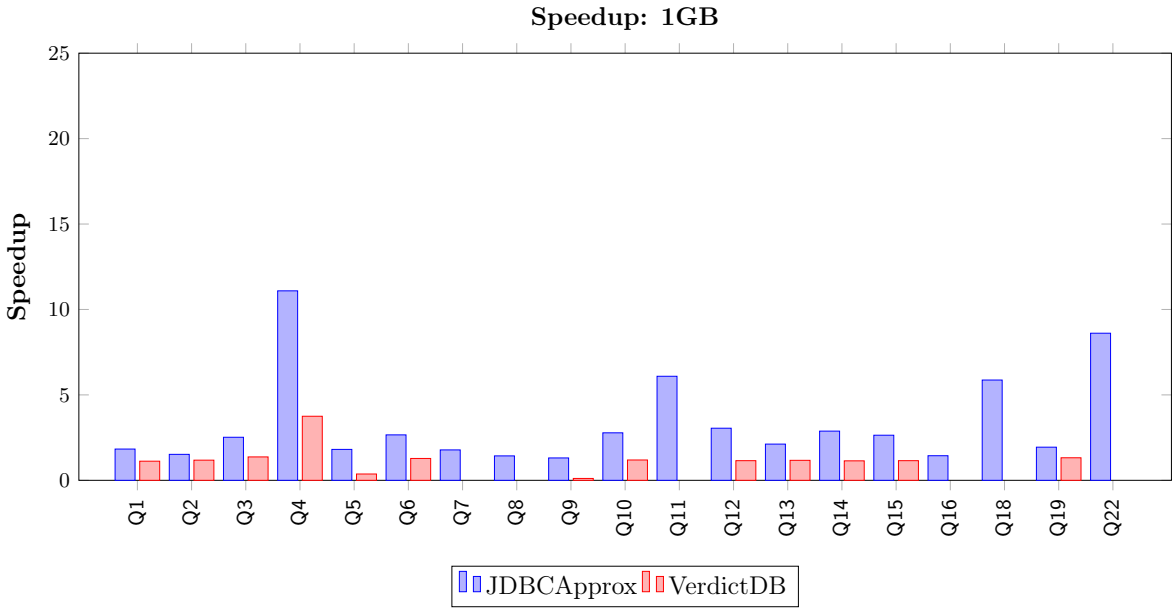

Figura 5.37: *Speedup* para cada pesquisa com 1GB de dados

Com a execução das pesquisas Q5 e Q9 através do VerdictDB não é obtido nenhum *speedup*, uma vez que estas utilizam seis tabelas e um número elevado de restrições na cláusula *Where*. Com o alto grau de complexidade destas pesquisas, a sua execução alocava muita memória RAM, sendo mais lenta em relação ao PostgreSQL.
O *speedup* de cerca de 11 vezes na pesquisa Q4 está relacionado com o facto da amostra da tabela *Order* ser diretamente criada a partir da amostra da tabela *Lineitem*, não existindo mais dados para além dos que participam na junção destas duas. Além disso, o PostgreSQL apresenta maiores tempos de execução com sub-pesquisas correlacionadas.
O *speedup* médio para as pesquisas que são executadas no VerdictDB e no JDBCApprox, com 1GB de dados é de 3 vezes no JDBCApprox e de 1.25 vezes no VerdictDB.
O principal motivo para o *speedup* com o JDBCApprox ser superior é a utilização de uma base de dados em memória. O desempenho deste tipo de configuração da base de dados é melhor, uma vez que ler e gravar dados em memória é mais rápido do que efetuar as mesmas operações em disco.
O gráfico da figura 5.38 ilustra o *speedup* para as pesquisas executadas no JDBCApprox e no VerdictDB para 10GB.

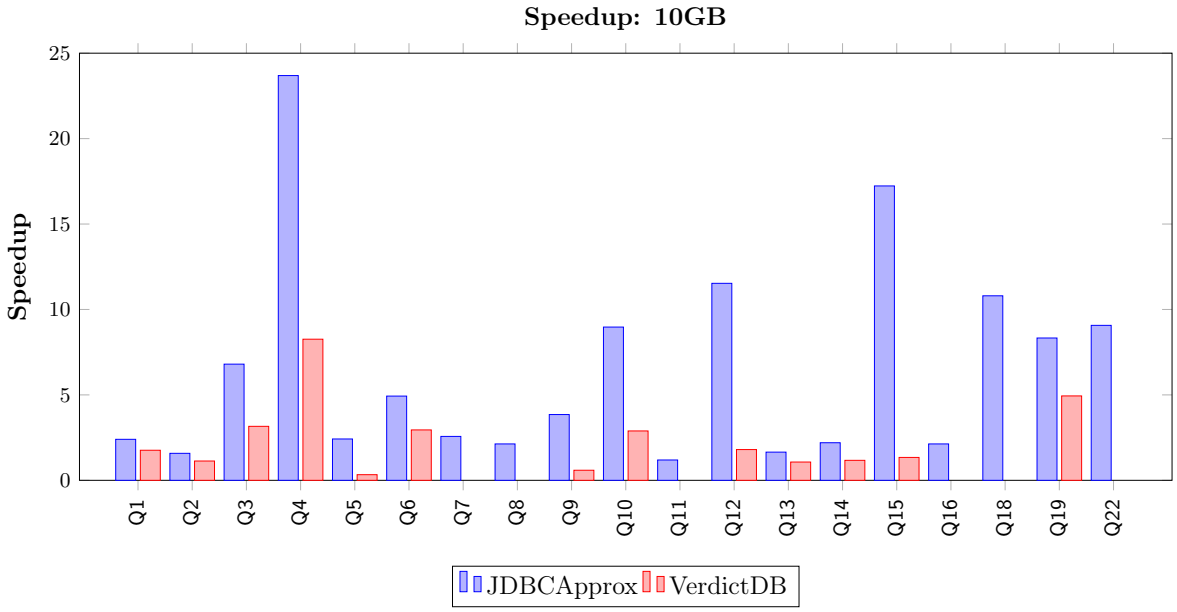

Figura 5.38: *Speedup* para cada pesquisa com 10GB de dados

À semelhança dos resultados anteriores para o *speedup* alcançado com 1GB de dados, a pesquisa Q4 mais uma vez consegue um *speedup* elevado, de cerca de 24 vezes com 10GB de dados utilizando o JDBCApprox. Este valor também poderá estar relacionado com o facto da pesquisa Q4 realizar uma sub-pesquisa, onde posteriormente armazena o resultado desta numa variável e realiza a pesquisa com o resultado obtido. Como é utilizada uma base de dados com configuração em memória, todas estas operações apresentam um melhor desempenho.
As pesquisas Q5 e Q9 voltam a não alcançar nenhum *speedup* com o VerdictDB, uma vez que é alocada muita memória RAM para obter as suas respostas. Os dados da tabela de amostra são divididos em blocos, para posteriormente serem executados bloco a bloco, através da execução da mesma pesquisa utilizando diversos blocos em simultâneo. Pelo mesmo motivo o *speedup* do VerdictDB começa a ser pouco percetível com o aumento da quantidade de dados.
O *speedup* médio para as pesquisas que são executadas no VerdictDB e no JDBCApprox, com 10GB de dados é de 7.35 vezes no JDBCApprox e de 2.41 vezes no VerdictDB.

5.4 Conclusão da avaliação experimental

A abordagem proposta na maioria das vezes apresenta respostas com um erro relativo inferior ao VerdictDB, quando se utilizam pesquisas com a cláusula GROUP BY que não envolva dados muito específicos (por exemplo, chaves primárias).
Em relação ao *speedup*, a abordagem proposta é sempre mais rápida, devido à utilização de uma base de dados em memória, que elimina a necessidade de aceder aos dados em disco. Além disso, a quantidade de dados que JDBCApprox utiliza para responder de forma aproximada às pesquisas é menor do que quantidade utilizada pelo VerdictDB. O JDBCApprox consegue ser 24 vezes mais rápido com 10GB de dados.

Capítulo 6

Conclusão

Durante o primeiro semestre foi realizado o estudo do estado da arte em sistemas de processamento aproximado de pesquisas, que permitiu conhecer quais são as ferramentas de AQP mais recentes e identificar as suas limitações. Inicialmente foram estudados os conceitos que estão na base do processamento aproximado de *queries*, sendo analisadas as suas duas abordagens: (i) agregação online e (ii) geração offline de sinopses. Em seguida foram estudadas as técnicas mais recentes de AQP, sendo apresentado o seu modo de funcionamento e as suas vantagens e desvantagens. No final, foi realizada uma comparação entre essas técnicas e descritas as suas limitações.
No segundo semestre foi desenvolvida uma biblioteca Java para mitigar algumas das limitações dos atuais sistemas de AQP: (i) não requer que seja efetuada qualquer alteração na base de dados, possuindo uma arquitetura de *middleware*; (ii) possui uma interface, que permite que os utilizadores compreendam facilmente os resultados devolvidos; (iii) é possível parametrizar o grau de confiança e o erro máximo admitido e (iv) lida com pesquisas que utilizem mais do que uma vez a mesma tabela. Esta biblioteca utiliza uma amostragem aleatória simples sem repetição para construir amostras do dados e em seguida utiliza uma base de dados com configuração em memória para acelerar a execução das pesquisas.
Através da utilização da amostragem aleatória simples sem repetição para criar amostras, sendo estas posteriormente copiadas para uma base de dados com configuração em memória, foi possível construir um sistema com uma arquitetura de *middleware*, que lida com a maioria do tipo de pesquisas. Com a possibilidade de parametrização do grau de confiança e do erro máximo admitido para a resposta de uma pesquisa, os utilizadores conseguem entender o quanto uma resposta é exata. Além disso, com a construção de uma interface para o utilizador, este consegue entender mais facilmente como funciona o JDBCApprox e quais as respostas que este devolve.
A biblioteca desenvolvida apresenta resultados mais exatos do que as abordagens existentes e consegue ser 24 vezes mais rápida, sendo na maior parte dos casos duas a três vezes mais rápida.

6.1 Trabalho futuro

Como trabalho futuro propomos que sejam realizados testes com uma maior quantidade de dados e com diferentes conjuntos de dados, de modo a perceber como a abordagem proposta se comporta em diferentes condições. Propomos também a utilização de uma técnica diferente quando são utilizadas pesquisas com junções, para serem atingidos melhores resultados quando a quantidade de registos em cada grupo é muito diferente.

6.2 Contribuições

O trabalho elaborado no âmbito desta dissertação resultou em diversos contributos.
Inicialmente, do ponto de vista académico, através da análise do estado da arte e da aquisição de conhecimento que foi possível com a investigação realizada. Desta investigação resultou o artigo "Evaluation of Approximate Query Processing Systems", publicado na conferência *LACCEI International Multi-Conference for Engineering, Education and Technology*. Este artigo consistiu na seleção de duas ferramentas de AQP que apresentavam os melhores resultados: VerdictDB e XDB, sendo posteriormente comparadas em termos de *speedup* e das taxas de erro que devolviam quando eram realizadas *queries*. Este artigo permitiu conhecer melhor estas duas técnicas, identificando as suas diferenças e limitações.
Segundo, através da biblioteca desenvolvida foi possível lidar com o processamento aproximado de pesquisas e entender quais as etapas necessárias para o seu bom funcionamento. Esta biblioteca permite devolver aos utilizadores respostas aproximadas de forma rápida e com uma taxa de erro reduzida. Além disso, permite aos programadores utilizarem a API e efetuarem alterações na mesma.
Adicionalmente, do estudo realizado durante esta dissertação, resultou o artigo "*JDBCApprox: an approximate processing system for in-memory queries*", a submeter a uma revista. O artigo focou-se essencialmente nos detalhes da abordagem proposta e nos resultados obtidos durante a avaliação experimental.

Referências

[1] Approxjoin approximate distributed joins. URL: https://approxjoin.github.io/.

[2] Blinkdb: Queries with bounded errors and bounded response times on very large data. URL: http://blinkdb.org/.

[3] Data never sleeps 6.0. URL: https://www.domo.com/learn/data-never-sleeps-6.

[4] Gaussian distribution function. URL: http://hyperphysics.phy-astr.gsu.edu/hbase/Math/gaufcn.html.

[5] Gpu ssd accelerates postgresql. URL: http://slideshare.net/kaigai/gpussd-accelerates-postgresql-challenge-towards-query-processing-throughput-10gbs.

[6] List of in-memory databases. URL: https://www.baeldung.com/java-in-memory-databases.

[7] postgresql-to-sqlite (pg2sqlite). URL: https://github.com/caiiiycuk/postgresql-to-sqlite.

[8] Qual é o tamanho da amostra que eu preciso? URL: https://www.netquest.com/blog/br/blog/br/qual-e-o-tamanho-de-amostra-que-preciso.

[9] Speedment. URL: https://www.speedment.com/about/.

[10] Speedup - wikipedia. URL: https://en.wikipedia.org/wiki/Speedup.

[11] Sprincípio da localidade. URL: http://producao.virtual.ufpb.br/books/edusantana/introducao-a-arquitetura-de-computadores-livro/livro/livro.chunked/ch05s02.html.

[12] Top 10 databases for 2019. URL: https://www.databasejournal.com/features/oracle/slideshows/top-10-2019-databases.html.

[13] Tpc-h. URL: http://www.tpc.org/tpch/.

[14] Swarup Acharya, Phillip B. Gibbons, and Viswanath Poosala. Aqua: A fast decision support systems using approximate query answers. In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 754–757, 1999. URL: http://www.vldb.org/conf/1999/P76.pdf.

[15] Sameer Agarwal. *Queries with Bounded Errors & Bounded Response Times on Very Large Data*. PhD thesis, University of California, Berkeley, USA, 2014. URL: http://www.escholarship.org/uc/item/58m3199x.

[16] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In *Eighth Eurosys Conference 2013, EuroSys '13, Prague, Czech Republic, April 14-17, 2013*, pages 29–42, 2013. URL: https://doi.org/10.1145/2465351.2465355, doi:10.1145/2465351.2465355.

[17] K. Bakshi. Considerations for big data: Architecture and approach. *2012 IEEE Aerospace Conference*, 2012. doi:10.1109/aero.2012.6187357.

[18] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970. URL: https://doi.org/10.1145/362686.362692, doi:10.1145/362686.362692.

[19] Siyao Cheng, Jianzhong Li, Qianqian Ren, and Lei Yu. Bernoulli sampling based -approximate aggregation in large-scale sensor networks. In *Proceedings of the 29th Conference on Information Communications*, INFOCOM'10, pages 1181–1189, Piscataway, NJ, USA, 2010. IEEE Press. URL: http://dl.acm.org/citation.cfm?id=1833515.1833693.

[20] William G. Cochran. *Sampling Techniques, Third Edition*. John Wiley & Sons, 3rd edition, 1977.

[21] Graham Cormode, Minos N. Garofalakis, Peter J. Haas, and Chris Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends in Databases*, 4(1-3):1–294, 2012. URL: https://doi.org/10.1561/1900000004, doi:10.1561/1900000004.

[22] Gautam Das. Sampling methods in approximate query answering systems. In *Encyclopedia of Data Warehousing and Mining, Second Edition (4 Volumes)*, pages 1702–1707. 2009. URL: http://www.igi-global.com/Bookstore/Chapter.aspx?TitleId=11047.

[23] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008. URL: http://doi.acm.org/10.1145/1327452.1327492, doi:10.1145/1327452.1327492.

[24] Amr El Abbadi, Michael L. Brodie, Sharma Chakravarthy, Umeshwar Dayal, Nabil Kamel, Gunter Schlageter, and Kyu-Young Whang, editors. *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*. Morgan Kaufmann, 2000.

[25] Zoubin Ghahramani. *Learning dynamic Bayesian networks*, pages 168–197. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998. URL: https://doi.org/10.1007/BFb0053999, doi:10.1007/BFb0053999.

[26] Iñigo Goiri, Ricardo Bianchini, Santosh Nagarakatte, and Thu D. Nguyen. Approxhadoop: Bringing approximations to mapreduce frameworks. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '15, Istanbul, Turkey, March 14-18, 2015*, pages 383–397, 2015. URL: https://doi.org/10.1145/2694344.2694351, doi:10.1145/2694344.2694351.

[27] A. Graps. An introduction to wavelets. *IEEE Computational Science and Engineering*, 2(2):50–61, 1995. doi:10.1109/99.388960.

[28] Sudipto Guha and Boulos Harb. Wavelet synopsis for data streams: minimizing non-euclidean error. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, Illinois, USA, August 21-24, 2005*, pages 88–97, 2005. URL: https://doi.org/10.1145/1081870.1081884, doi:10.1145/1081870.1081884.

[29] Silviu Guiasu and Abe Shenitzer. The principle of maximum entropy. *The Mathematical Intelligencer*, 7(1):42–48, Mar 1985. URL: https://doi.org/10.1007/BF03023004, doi:10.1007/BF03023004.

[30] Thomas Heinis and David A. Ham. On-the-fly data synopses: Efficient data exploration in the simulation sciences. *SIGMOD Record*, 44(2):23–28, 2015. URL: https://doi.org/10.1145/2814710.2814715, doi:10.1145/2814710.2814715.

[31] Joseph M. Hellerstein, Peter J. Haas, and Helen J. Wang. Online aggregation. In *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA.*, pages 171–182, 1997. URL: https://doi.org/10.1145/253260.253291, doi:10.1145/253260.253291.

[32] Barbara Hołyńska. Sampling and sample preparation in edxrs. *X-Ray Spectrometry*, 22(4):192–198, 1993. doi:10.1002/xrs.1300220406.

[33] Tomohiro Inoue. *An effective technique and practical utility for approximate query processing.* PhD thesis, Curtin University, 2015.

[34] Donald E. Knuth. *The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms.* Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1997.

[35] Feifei Li, Bin Wu, Ke Yi, and Zhuoyue Zhao. Wander join and XDB: online aggregation via random walks. *SIGMOD Record*, 46(1):33–40, 2017. URL: https://doi.org/10.1145/3093754.3093763, doi:10.1145/3093754.3093763.

[36] Kaiyu Li and Guoliang Li. Approximate query processing: What is new and where to go? - A survey on approximate query processing. *Data Science and Engineering*, 3(4):379–397, 2018. URL: https://doi.org/10.1007/s41019-018-0074-4, doi:10.1007/s41019-018-0074-4.

[37] Qingwei Lin, Weichen Ke, Jian-Guang Lou, Hongyu Zhang, Kaixin Sui, Yong Xu, Ziyi Zhou, Bo Qiao, and Dongmei Zhang. Bigin4: Instant, interactive insight identification for multi-dimensional big data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, pages 547–555, 2018. URL: https://doi.org/10.1145/3219819.3219867, doi:10.1145/3219819.3219867.

[38] Qing Liu. Approximate query processing. In *Encyclopedia of Database Systems*, pages 113–119. 2009. URL: https://doi.org/10.1007/978-0-387-39940-9_534, doi:10.1007/978-0-387-39940-9_534.

[39] Dominik Moritz and Danyel Fisher. What users don't expect about exploratory data analysis on approximate query processing systems. In *Proceedings of the 2nd Workshop on Human-In-the-Loop Data Analytics, HILDA@SIGMOD 2017, Chicago, IL, USA, May 14, 2017*, pages 9:1–9:4, 2017. URL: https://doi.org/10.1145/3077257.3077258, doi:10.1145/3077257.3077258.

[40] Barzan Mozafari. Approximate query engines: Commercial challenges and research opportunities. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, pages 521–524, 2017. URL: https://doi.org/10.1145/3035918.3056098, doi:10.1145/3035918.3056098.

[41] Barzan Mozafari, Jags Ramnarayan, Sudhir Menon, Yogesh Mahajan, Soubhik Chakraborty, Hemant Bhanawat, and Kishor Bachhav. Snappydata: A unified cluster for streaming, transactions and interactice analytics. In *CIDR 2017, 8th Biennial Conference on Innovative Data Systems Research, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings*, 2017. URL: http://cidrdb.org/cidr2017/papers/p28-mozafari-cidr17.pdf.

[42] Frank Olken and Doron Rotem. Random sampling from databases - a survey. *Statistics and Computing*, 5:25–42, 1994.

[43] Fatma Özcan, Georgia Koutrika, and Sam Madden, editors. *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*. ACM, 2016. URL: https://doi.org/10.1145/2882903, doi:10.1145/2882903.

[44] Yongjoo Park, Barzan Mozafari, Joseph Sorenson, and Junhao Wang. Verdictdb: Universalizing approximate query processing. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 1461–1476, 2018. URL: https://doi.org/10.1145/3183713.3196905, doi:10.1145/3183713.3196905.

[45] Yongjoo Park, Ahmad Shahab Tajik, Michael J. Cafarella, and Barzan Mozafari. Database learning: Toward a database that becomes smarter every time. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, pages 587–602, 2017. URL: https://doi.org/10.1145/3035918.3064013, doi:10.1145/3035918.3064013.

[46] Gregory Piatetsky-Shapiro and Charles Connell. Accurate estimation of the number of tuples satisfying a condition. In *SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, USA, June 18-21, 1984*, pages 256–276, 1984. URL: https://doi.org/10.1145/602259.602294, doi:10.1145/602259.602294.

[47] Do Le Quoc. *Approximate Data Analytics Systems*. PhD thesis, Technische Universität Dresden, 2017.

[48] Do Le Quoc, Istemi Ekin Akkus, Pramod Bhatotia, Spyros Blanas, Ruichuan Chen, Christof Fetzer, and Thorsten Strufe. Approxjoin: Approximate distributed joins. In *Proceedings of the ACM Symposium on Cloud Computing, SoCC 2018, Carlsbad, CA, USA, October 11-13, 2018*, pages 426–438, 2018. URL: https://doi.org/10.1145/3267809.3267834, doi:10.1145/3267809.3267834.

[49] Do Le Quoc, Ruichuan Chen, Pramod Bhatotia, Christof Fetzer, Volker Hilt, and Thorsten Strufe. Streamapprox: approximate computing for stream analytics. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference, Las Vegas, NV, USA, December 11 - 15, 2017*, pages 185–197, 2017. URL: https://doi.org/10.1145/3135974.3135989, doi:10.1145/3135974.3135989.

[50] Hamed Taherdoost. Sampling methods in research methodology; how to choose a sampling technique for research. *SSRN Electronic Journal*, 2016. doi:10.2139/ssrn.3205035.

[51] Steven K. Thompson. Sampling. *Wiley Series in Probability and Statistics*, 2012. doi:10.1002/9781118162934.

[52] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. Hive - A warehousing solution over a map-reduce framework. *PVLDB*, 2(2):1626–1629, 2009. URL: http://www.vldb.org/pvldb/2/vldb09-938.pdf, doi:10.14778/1687553.1687609.

[53] Yuxiang Wang, Yixing Xia, Qiming Fang, and Xiaoliang Xu. AQP++: a hybrid approximate query processing framework for generalized aggregation queries. *J. Comput. Science*, 26:419–431, 2018. URL: https://doi.org/10.1016/j.jocs.2017.05.001, doi:10.1016/j.jocs.2017.05.001.

[54] Yuxiang Wang, Xiaoliang Xu, Yixing Xia, and Qiming Fang. AQP++: A hybrid approximate query processing framework for generalized aggregation queries. In *International Conference on Advanced Cloud and Big Data, CBD 2016, Chengdu, China, August 13-16, 2016*, pages 56–62, 2016. URL: https://doi.org/10.1109/CBD.2016.020, doi:10.1109/CBD.2016.020.

[55] Sai Wu, Beng Chin Ooi, and Kian-Lee Tan. Online aggregation. In *Advanced Query Processing, Volume 1: Issues and Trends*, pages 187–210. 2013. URL: https://doi.org/10.1007/978-3-642-28323-9_8, doi:10.1007/978-3-642-28323-9_8.

[56] Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, and Ion Stoica. Discretized streams: fault-tolerant streaming computation at scale. In *ACM SIGOPS 24th Symposium on Operating Systems Principles, SOSP '13, Farmington, PA, USA, November 3-6, 2013*, pages 423–438, 2013. URL: https://doi.org/10.1145/2517349.2522737, doi:10.1145/2517349.2522737.

Anexo A

Criação de Amostras

Este anexo apresenta o código SQL utilizado na criação das amostras de cada tabela do *benchmark* TPC-H.

```
create table lineitem_sample as select * from lineitem where random()<=0.2;
```

```
create table partsupp_sample as select distinct ps_partkey, ps_suppkey, ps_availqty,
ps_supplycost, ps_comment from partsupp inner join lineitem_sample on ps_partkey
= l_partkey and ps_suppkey =l_suppkey;
```

```
create table supplier_sample as select distinct s_suppkey, s_name, s_address,
s_nationkey, s_phone, s_acctbal, s_comment from supplier inner join partsupp_sample
on s_suppkey = ps_suppkey;
```

```
create table orders_sample as select distinct o_orderkey, o_custkey, o_orderstatus,
o_totalprice, o_orderdate, o_orderpriority, o_clerk, o_shippriority, o_comment
from orders join lineitem_sample on o_orderkey = l_orderkey;
```

```
create table customer_sample as select distinct c_custkey, c_name, c_address,
c_nationkey, c_phone, c_acctbal, c_mktsegment, c_comment from customer inner
join orders_sample on c_custkey =o_custkey;
```

```
create table nation_sample as select distinct n_nationkey, n_name, n_regionkey,
n_comment from nation inner join customer_sample on n_nationkey = c_nationkey;
```

```
create table region_sample as select distinct r_regionkey, r_name, r_comment
from region inner join nation_sample on r_regionkey = n_regionkey;
```

```
create table part_sample as select distinct p_partkey, p_name, p_mfgr, p_brand,
p_type, p_size, p_container, p_retailprice, p_comment from part inner join
partsupp_sample on p_partkey= ps_partkey;
```

Anexo B

Pesquisas padrão do *benchmark* TPC-H

Este anexo apresenta o código SQL das pesquisas do *benchmark* TPC-H utilizadas nas experiências. Para cada uma das pesquisas é fornecida uma breve explicação do seu funcionamento.

Pesquisa Q1: Relatório do resumo de preços
Essa pesquisa informa os números do negócio relativos a quantidades faturadas, expedidas e devolvidas. A pesquisa relatório do resumo de preços fornece um relatório resumido dos preços para todos os artigos expedidos a partir de uma determinada data. Esta data está contida num intervalo de 60 a 120 dias da data do último envio presente na base de dados. Nesta pesquisa é realizada uma leitura de mais de 95% dos registos da maior tabela da base de dados.

```
select
    l_returnflag,
    l_linestatus,
    sum(l_quantity) as sum_qty,
    sum(l_extendedprice) as sum_base_price,
    sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
    sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
    avg(l_quantity) as avg_qty,
    avg(l_extendedprice) as avg_price,
    avg(l_discount) as avg_disc,
    count(*) as count_order
from
    lineitem
where
    l_shipdate <= date ('1998-12-01', 'YYYY-MM-DD') - interval '90' day
group by
    l_returnflag,
    l_linestatus
order by
    l_returnflag,
    l_linestatus;
```

Pesquisa Q2: Fornecedor de custo mínimo
Esta pesquisa identifica qual fornecedor deve ser selecionado para fazer um pedido para uma determinada parte de uma determinada região. A pesquisa de fornecedor de custo mínimo localiza, para cada parte de um determinado tipo e tamanho, qual o fornecedor que a possa disponibilizar a um custo mínimo. Para facilitar a análise de resultados esta pesquisa foi alterada, de modo a devolver apenas cinco fornecedores.

```
select
    sum(s_acctbal) as acctbal,
    n_name
from
    part,
    supplier,
    partsupp,
    nation,
    region
where
    p_partkey = ps_partkey
    and s_suppkey = ps_suppkey
    and p_size = 15
    and p_type like '%BRASS'
    and s_nationkey = n_nationkey
    and n_regionkey = r_regionkey
    and r_name = 'EUROPE'
group by
    n_name
order by
    acctbal,
    n_name
limit 5;
```

Pesquisa Q3: Prioridade de envio
Esta pesquisa lista os 10 pedidos não enviados com o valor mais alto. Nesta pesquisa são devolvidas a prioridade de envio e a potencial receita, dos pedidos com a maior receita entre aqueles que não foram enviados a partir de uma determinada data. Para facilitar a análise de resultados esta pesquisa foi alterada, de modo a devolver apenas cinco pedidos.

```
select
    l_orderkey,
    sum(l_extendedprice*(1-l_discount)) as revenue,
    o_orderdate,
    o_shippriority
from
    customer,
    orders,
    lineitem
where
    c_mktsegment in ('BUILDING', 'AUTOMOBILE')
    and c_custkey = o_custkey
    and l_orderkey = o_orderkey
```

```
    and o_orderdate < date '1995-03-15'
    and l_shipdate > '1995-03-15'
group by
    l_orderkey,
    o_orderdate,
    o_shippriority
order by
    revenue desc,
    o_orderdate
limit 5;
```

Pesquisa Q4: Verificação do sistema de controlo de prioridade de pedidos
Esta pesquisa verifica a eficácia do sistema de controlo de prioridade de pedidos e avalia o nível de satisfação do cliente. Nesta pesquisa o número de pedidos são ordenados num determinado trimestre de um determinado ano, onde pelo menos um pedido foi recebido pelo cliente após a data de entrega.

```
select
    o_orderpriority,
    count(*) as order_count
from
    orders
where
    o_orderdate >= date '1993-10-01'
    and o_orderdate < date '1993-10-01' + interval '3' month
    and exists(
            select
                *
            from
                lineitem
            where
                l_orderkey = o_orderkey
                and l_commitdate < l_receiptdate
    )
group by
    o_orderpriority
order by
    o_orderpriority;
```

Pesquisa Q5: Volume de receitas através de fornecedores locais
Esta pesquisa lista, para cada nação numa determinada região, o volume de receitas resultante de transações de produtos onde o cliente que solicitou as peças e o fornecedor estavam na mesma nação. A pesquisa é executada para determinar se é necessário instituir centros de distribuição locais numa determinada região, considerando apenas peças encomendadas num determinado ano.

```
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 = 'ASIA'
    and o_orderdate >= date '1994-01-01'
    and o_orderdate < date ' 1994-01-01' + interval '1' year
group by
    n_name
order by
    revenue desc;
```

Pesquisa Q6: Previsão dos lucros com alterações
Esta pesquisa quantifica o aumento no lucro que teria resultado da eliminação de determinados descontos em toda a empresa, num determinado intervalo percentagem, num determinado ano. O resultado desta pesquisa pode ser utilizado para estudar alternativas para aumentar os lucros.

```
select
    sum(l_extendedprice*l_discount) as revenue
from
    lineitem
where
    l_shipdate >= date '1994-01-01'
    and l_shipdate < date '1994-01-01' + interval '1' year
    and l_discount between 0.06 - 0.01 and 0.06 + 0.01
    and l_quantity < 24;
```

Pesquisa Q7: Volume de vendas
Esta pesquisa determina o valor das mercadorias enviadas entre determinadas nações para auxiliar na renegociação dos contratos de vendas.

```
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 = 'FRANCE' and n2.n_name = 'GERMANY')
                    or (n1.n_name = 'GERMANY' and n2.n_name = 'FRANCE')
                    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;
```

Pesquisa Q8: Quota de mercado
Esta pesquisa determina como a quota de mercado de uma determinada nação dentro de uma determinada região mudou ao longo de dois anos para um determinado tipo de peça.

```
select
    o_year,
    sum(case
            when nation = 'BRAZIL'
            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 = 'AMERICA'
            and s_nationkey = n2.n_nationkey
            and o_orderdate between date '1995-01-01' and date '1996-12-31'
            and p_type = 'ECONOMY ANODIZED STEEL'
    ) as all_nations
group by
    o_year
order by
    o_year;
```

Pesquisa Q9: Lucro numa determinada linha de peças
Esta pesquisa determina o lucro feiro sobre uma determinada linha de peças, divididas por nação, fornecedor e ano. O tipo de produto procura, para cada nação e cada ano, o lucro de todas as peças encomendadas naquele ano que possuem uma *substring* específica e que foram preenchidas por um fornecedor naquela nação. Para facilitar a análise de resultados esta pesquisa foi alterada, de modo a devolver apenas cinco linhas.

```
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
amount
        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 '%green%'
            ) as profit
group by
    nation,
    o_year
order by
    nation,
    o_year desc
limit 5;
```

Pesquisa Q10: Relatório de produtos devolvidos
Esta pesquisa identifica clientes que possam estar a ter problemas com as peças que lhes são enviadas. Nesta pesquisa são encontrados os 20 principais clientes, em termos do seu efeito na receita perdida num determinado trimestre, que devolveram produtos. Para facilitar a análise de resultados esta pesquisa foi alterada, de modo a devolver apenas os cinco principais clientes.

```
select
    c_custkey,
    c_name,
    sum(l_extendedprice) as revenue,
    c_acctbal
from
    customer,
    orders,
    lineitem,
    nation
where
    c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and o_orderdate >= date '1993-10-01'
    and o_orderdate < date '1993-10-01' + interval '3' month
    and l_returnflag = 'R'
    and c_nationkey = n_nationkey
group by
    c_custkey,
    c_name,
    c_acctbal
order by
    c_acctbal desc
limit 5;
```

Pesquisa Q11: Identificação de *stock* importante
Esta pesquisa localiza o subconjunto mais importante de *stock* de fornecedores numa determinada nação. Nesta pesquisa são consideradas todas as partes que representem uma

percentagem significativa do valor total de todas as peças disponíveis, a partir da análise do *stock* disponível nos fornecedores de uma determinada nação. Para facilitar a análise de resultados esta pesquisa foi alterada, de modo a devolver apenas cinco linhas.

```
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 = 'GERMANY'
group by
    ps_partkey having
            sum(ps_supplycost * ps_availqty) > (
                                    select
                                        sum(ps_supplycost * ps_availqty) * 0.00001
                                    from
                                        partsupp,
                                        supplier,
                                        nation
                                    where
                                        ps_suppkey = s_suppkey
                                        and s_nationkey = n_nationkey
                                        and n_name = 'GERMANY'
                                        )
order by
    value desc
limit 5;
```

Pesquisa Q12: Métodos de envio e prioridade dos pedidos
Esta pesquisa determina se a seleção dos métodos de envio mais baratos está a afetar negativamente os pedidos de prioridade crítica, fazendo com que uma maior quantidade de peças sejam recebidas pelos clientes após a data de entrega prevista.

```
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 ('MAIL', 'SHIP')
    and l_commitdate < l_receiptdate
    and l_shipdate < l_commitdate
    and l_receiptdate >= date ' 1994-01-01'
    and l_receiptdate < date ' 1994-01-01' + interval '1' year
group by
    l_shipmode
order by
    l_shipmode;
```

Pesquisa Q13: Distribuição de clientes
Esta pesquisa procura obter relacionamentos entre clientes e o tamanho dos seus pedidos. É determinada a distribuição de clientes pelo total de pedidos que eles realizaram, incluindo clientes que não têm registo de pedidos, passados ou presentes.

```
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 '%special%requests%'
    group by
         c_custkey
    )as c_orders (c_custkey, c_count)
group by
    c_count
order by
    custdist desc,
    c_count desc;
```

Pesquisa Q14: Efeito promoção
Esta pesquisa monitoriza a resposta do mercado a uma promoção, como anúncios de TV ou uma campanha especial. Nesta pesquisa é determinada qual a percentagem da receita num determinado ano e mês que derivou de partes promocionais.

```
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 '1995-09-01'
    and l_shipdate < date '1995-09-01' + interval '1' month;
```

Pesquisa Q15: Principais fornecedores
Esta pesquisa determina os principais fornecedores para que estes possam ser recompensados, receberem mais negócios ou serem identificados para reconhecimento especial. Nesta pesquisa é encontrado o fornecedor que mais contribuiu para a receita total das peças enviadas durante um determinado trimestre de um determinado ano. Esta pesquisa foi alterada, uma vez que a pesquisa original consista na criação de uma vista.

```
select
    s_suppkey,
    s_name,
    s_address,
    s_phone,
    max(aux.total_revenue) as total
from
    supplier,
    (select
        l_suppkey,
        sum(l_extendedprice * (1 - l_discount)) as total_revenue
    from
        lineitem
    where
        l_shipdate >= date ' 1996-01-01'
        and l_shipdate < date ' 1996-01-01' + interval '3' month
    group by
        l_suppkey) as aux
where
    s_suppkey = aux.l_suppkey
group by
    s_suppkey,
    s_name,
    s_address,
    s_phone
order by
    total desc
limit 1;
```

Pesquisa Q16: Relação peças/fornecedores
Esta pesquisa identifica quantos fornecedores podem fornecer peças com determinadas caraterísticas. Pode ser usado, por exemplo, para determinar se há um número suficiente

de fornecedores para peças que normalmente não são encomendadas. Nesta pesquisa é calculado o número de fornecedores que podem fornecer peças que satisfazem os requisitos de um determinado cliente. Para facilitar a análise de resultados esta pesquisa foi alterada, de modo a devolver apenas cinco linhas.

```
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#45'
    and p_type not like 'MEDIUM POLISHED%'
    and p_size in (49, 14, 23, 45, 19, 3, 36, 9)
    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
limit 5;
```

Pesquisa Q17: Lucro de pedidos pequenos

Essa pesquisa determina qual seria a receita média anual perdida se os pedidos não fossem satisfeitos para pequenas quantidades de determinadas peças. Nesta pesquisa são consideradas as partes de uma determinada marca, com um determinado tipo de embalagem, sendo determinada a quantidade média de produtos solicitados para todos os pedidos.

```
select
    sum(l_extendedprice) / 7.0 as avg_yearly
from
    lineitem,
    part
where
    p_partkey = l_partkey
```

```
    and p_brand = 'Brand#23'
    and p_container = 'MED BOX'
    and l_quantity < (
                        select
                             0.2 * avg(l_quantity)
                        from
                             lineitem
                        where
                             l_partkey = p_partkey
                        );
```

Pesquisa Q18: Cliente de grande volume
Esta pesquisa classifica os clientes com base no facto de eles terem realizado um pedido de grande quantidade, onde a quantidade total está acima de um determinado nível. Para facilitar a análise de resultados esta pesquisa foi alterada, de modo a devolver apenas cinco clientes.

```
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) > 300
    )
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
limit 5;
```

Pesquisa Q19: Lucro com desconto
Esta pesquisa informa qual a receita bruta descontada atribuída à venda de peças selecionadas tratadas de uma maneira específica. Nesta pesquisa é identificada qual a receita bruta descontada para todos os pedidos para três tipos diferentes de peças, que foram enviadas por via aérea e entregues pessoalmente.

```
select
    sum(l_extendedprice * (1 - l_discount) ) as revenue
from
    lineitem,
    part
where
    (
    p_partkey = l_partkey
    and p_brand = 'Brand#12'
    and p_container in ( 'SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
    and l_quantity >= 1 and l_quantity <= 1 + 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 = 'Brand#23'
    and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
    and l_quantity >=10 and l_quantity <= 10 + 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 = 'Brand#34'
    and p_container in ( 'LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
    and l_quantity >= 20 and l_quantity <= 20 + 10
    and p_size between 1 and 15
    and l_shipmode in ('AIR', 'AIR REG')
    and l_shipinstruct = 'DELIVER IN PERSON'
    );
```

Pesquisa Q20: Possível promoção de uma peça
Esta pesquisa identifica fornecedores numa determinada nação, tendo em conta peças que podem ser candidatas a uma oferta promocional. Nesta pesquisa são identificados os fornecedores que possuem um excedente de uma determinada peça disponível.

```
select
    s_name,
    s_address
```

```
from
    supplier,
    nation
where
    s_suppkey in (
                select
                     ps_suppkey
                from
                    partsupp
                where
                    ps_partkey in (
                                select
                                     p_partkey
                                from
                                     part
                                where
                                    p_name like 'forest%'
                    )
    and ps_availqty > (
                    select
                         0.5 * sum(l_quantity)
                    from
                         lineitem
                    where
                        l_partkey = ps_partkey
                        and l_suppkey = ps_suppkey
                        and l_shipdate >= date('1994-01-01')
                        and l_shipdate < date('1994-01-01') + interval '1' year
                    )
    )
    and s_nationkey = n_nationkey
    and n_name = 'CANADA'
order by
    s_name;
```

Pesquisa Q21: Fornecedores que mantiveram pedidos em espera
Esta pesquisa identifica os fornecedores que não conseguiram enviar as peças necessárias dentro de um período estipulado. Nesta pesquisa os fornecedores são identificados para uma determinada nação.

```
select
    s_name,
    count(*) as numwait
from
    supplier,
    lineitem l1,
    orders,
    nation
where
    s_suppkey = l1.l_suppkey
```

```
    and o_orderkey = l1.l_orderkey
    and o_orderstatus = 'F'
    and l1.l_receiptdate > l1.l_commitdate
    and exists (
            select
                    *
            from
                lineitem l2
            where
                l2.l_orderkey = l1.l_orderkey
                and l2.l_suppkey <> l1.l_suppkey
            )
    and not exists (
                 select
                          *
                from
                     lineitem l3
                where
                    l3.l_orderkey = l1.l_orderkey
                    and l3.l_suppkey <> l1.l_suppkey
                    and l3.l_receiptdate > l3.l_commitdate
                    )
    and s_nationkey = n_nationkey
    and n_name = 'SAUDI ARABIA'
group by
    s_name
order by
    numwait desc,
    s_name;
```

Pesquisa Q22: Oportunidade de vendas globais

Esta pesquisa identifica locais onde há clientes que possam efetuar uma compra. Nesta pesquisa é contabilizado o número de clientes dentro de um intervalo específico de códigos de países que não fizeram compras durante 7 anos, mas têm um saldo de conta positivo, maior do que a média.

```
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 ('13','31','23','29','30','18','17')
            and c_acctbal > (
                select
```

```
                    avg(c_acctbal)
                from
                    customer
                where
                    c_acctbal > 0.00
                    and substring (c_phone from 1 for 2) in
                ('13','31','23','29','30','18','17')
                )
        and not exists (
                select
                        *
                from
                    orders
                where
                    o_custkey = c_custkey
                )
        ) as custsale
group by
    cntrycode
order by
    cntrycode;
```

Anexo C

Artigo Científico

Este anexo apresenta o artigo publicado na conferência "17th LACCEI International Multi-Conference for Engineering, Education, and Technology".

Evaluation of Approximate Query Processing Systems

Solange Paz[1], Bruno Cabral[1,2] and Jorge Bernardino[2,3]
[1]*Department of Informatics Engineering, University of Coimbra, Portugal*
[2]*CISUC, Centre for Informatics and Systems of University Coimbra, Portugal*
[3]*Department of Informatics and Systems Engineering, Polytechnic of Coimbra – ISEC, Coimbra, Portugal*
spaz@student.dei.uc.pt, bcabral@dei.uc.pt, jorge@isec.pt

Keywords: Approximate query processing, error rate, query response time.

Abstract: Today data-intensive systems, such as e-business. e-procurement e-government, e-commerce etc. systems present a huge amount of available data. One of the main issues of query processing is how to process queries efficiently. In many cases, it is impossible or too expensive for users to get exact answers in a short query response time. Approximate query processing (AQP) is an alternative way that returns approximate answer and is increasingly used, as millions of data are processed daily in a database. In this paper, we evaluate two of the latest AQP systems with the best results in the literature: VerdictDB and XDB. We test these systems according to the query response time and accuracy of results returned, using queries of the TPC-H benchmark with different sizes. The VerdictDB and XDB are good tools for large volumes of data. The experiments demonstrate that VerdictDB results can be 76x faster than MySQL. However, with the same query response time, XDB returns results with more accuracy.

1 INTRODUCTION

In recent years Approximate Query Processing (AQP) is increasingly used, since the social media, mobile devices, and wireless sensors continue to create massive data volumes [1]. In a large company where this data increases daily is necessary to get answers quickly and accurately, in order to allow better decision making. The time to execute large data is too long. Currently users are waiting too long to get accurate results in large-scale data. Some users consider this time unacceptable, since the productivity of them is inhibited by slow and expensive data interaction. The goal of an AQP system is returning an answer in short response time. Thus, an approximate aggregate with a specified error warranty is an option to improve the productivity [2]. The AQP system allows users to change the accuracy of the query by the speed with which a response is returned in large data sets and using complex aggregation queries [3].

The applications of decision support and data mining often turn to aggregate functions, such as "SUM" and "AVG" to formulate a query. As these queries are performed in large databases, the time consumed is very high. In this type of systems, the accuracy of the results of the queries is not so relevant, as far as users prefer to know an approximate response instead of waiting too long to get an exact answer [4]. Approximate answers in these systems allow users to analyse data quickly and effectively.

To analyse data quickly, it is necessary to get an answer as accurately as possible in a short period of time. This is possible using approximate query processing techniques, once that reduce the response time of queries on online support systems, when it is not required nor important to obtain a very precise answer. [5].

There are numerous techniques for processing approximate queries and can be categorized into two categories: online aggregation and offline synopses generation [6]. The AQP with random sampling as a basis is one of the most useful methods for the calculated of large quantities of data in databases efficiently [7]. The first generation of AQP was focused on online aggregation for simple OLAP queries. The second extended the scope to more complex workflows mainly by taking pre-calculated samples of costs, if most or all queries are beforehand known. The third generation cannot assume that most queries are known in advance, but instead can leverage that data exploration pipelines are incrementally created by the user with a visual interface [8].

For a large-scale data, a less accurate but instantaneous result is desirable. Query processing is the process that deduces information presented in the database. In many cases is impossible know how to process queries efficiently and obtain the exact answers as soon as possible [9]. The AQP consists in using a sample of the total data and processing queries with those sampled data.
The results of the approximate answer are better as more data is available and, if have time for the continue the processing the data converges for an exact answer [10]. The AQP is designed for aggregate queries such as using the AVERAGE, COUNT and SUM functions.
We selected two of latest tools of AQP with better results in the literature review: VerdictDB and the XDB. Using TPC-H benchmark queries, we compare these tools using different database sizes of 1, 10, and 50 GB. We also compared the tools with the original DBMSs that support them, in order to understand the differences in speedup and data accuracy.
The main contributions of this paper are as follows: analyse the differences in terms of performance and error rate (accuracy) between the two tools and compare the performance of the tools in relation to MySQL and PostgreSQL.
The rest of this paper is structured as follows. Section 2 presents a literature review on approximate query processing and Section 3 describes the experimental setup, included the results of the experiment. Finally, Section 4 states the conclusions and proposes some future work.

2 RELATED WORK

The sampling techniques from base relations in order to quickly estimate the answer to an aggregation query are the first that appear described in the research. However, these sampling techniques are not directly applicable to online aggregation. In online aggregation an execution estimate is continuously updated based on the data known up to now. The error in this estimate is specified through a confidence interval [11]. One of the first proposals was ripple join, an online aggregation interface that permits users to observe in real time the progress of their aggregation queries and control the execution [12]. The wander join chooses the optimal plan for conducting the random walks without having to use and collect any statistics initially. Compared with ripple join, the wander join is more efficient for joins [13].
The XDB [14] integrates the wander join in PostgreSQL and comparing XDB with Turbo DBO. The comparing was made when there is enough memory and when there is no enough memory.
In [15] the authors presented a BlinkDB allows users to trade off query accuracy for response time, since it allows interactive queries with over massive data. BlinkDB presented results with error rating, being 200x faster than Hive within an error of 2-10%.
The VerdictDB [16] makes all communications with the backend database in SQL and return a very fast result. The authors show how this tool is being in 171x faster than other existing engines, for example, Spark SQL.
In [17] the authors proposed a framework for creating and running approximation-enable MapReduce programs. The ApproxHadoop reduces application runtime and energy consumption, returning answers with very small errors. This framework reduces runtimes by up to 32x with an error of 1% and 95% confidence.
In [18] the authors proposed the use of multi-dimensional wavelets as an effective tool for general-purpose approximate query processing. They compared the wavelets with sampling techniques and histograms. The performance of histograms is worse than that of wavelets. Comparing histograms and sampling, wavelets exhibit more than an improvement on the magnitude of the relative error.
Unlike previous papers, besides a comparison between two recent tools for AQP, in our work it is also a comparison with these tools and the original DBMS that support them.

3 EXPERIMENTAL EVALUATION

In this section we present the results of the experimental study conducted to evaluate the two selected tools: VerdictDB and XDB. Using different data from the TPC-H benchmark show what the best tool. We start this section with a description of the tools evaluated. Then we describe the experimental scenario used, being then presented the results and conclusions of the respective evaluation.

3.1 Analysed Tools

We selected two recent AQP tools with better results in the literature review: VerdictDB [19] and XDB [20]. Although both tools provide approximate answers, they have differences and constraints. The XDB only support online aggregation and not support

the ORDER BY or LIMIT clause. In this tool is mandatory to define a confidence interval and the time we want to wait until we get a query result. But also, is possible visualizing various results and stop the query when a response very close to the original is shown. In contrast, the VerdictDB support the normal syntax of query, including also the ORDER BY or LIMIT.

VerdictDB [21] is an AQP system that uses a middleware architecture that does not require changes to the database and can work with all the data management systems available in the market. The user sends the queries to the VerdictDB and gets the result from them without interacting with the database. VerdictDB then communicates with the database to obtain metadata and to access and process data. If there is a set of sample tables that can be used in place of each of the base tables, the sample tables in the query are used. Each query is converted into logical operators and then converted to another logical expression capable of executing in the AQP, in the end it is rewritten in an SQL statement, to be executed. VerdictDB uses sub-sampling as a technique to estimate error through variational subsampling. In variational subsampling the same row of the table may belong to several subsamples and all subsamples must be the same size. For each row of the sub-sample, only a random number is generated to determine which sub-sample belongs to, and then the aggregation is performed only once per row. You define which tables you want to build samples of, which are usually those that contain a larger set of data. When a query is performed, the Query Parser converts it to logical expressions such as joins, which are then converted by the AQP Rewriter into another logical expression that can be executed with AQP. The Syntax Changer converts this rewritten logical expression into an SQL statement that can be executed in the underlying database. At the end, after this rewritten query is executed, Answer Rewriter returns an approximate response to the original query.

XDB is an engine developed into PostgreSQL 9.4.2 to support the wander join algorithm in online aggregation queries. The idea for wander join is to model a join over k tables as a join graph, and then perform random walks in this graph. In this graph, each tuple is modeled as a vertex and there is an edge between two tuples if they can join, are have the same value on their common attribute. How can they exist various random walks for the same query, XDB have a mechanism who chooses the best walking plan. This mechanism consists of the execution of a series of trial runs, using multiple paths in order to find the best. After selecting the best walk path, are extracted samples of B-tree indexes, one by one. The counts of the subtrees of the internal nodes of indices are used to extract samples. The selection predicates are applied when the related tuples are sampled. When the walk is complete, the executor returns the current estimators and confidence intervals. It returns an empty tuple when the time set in the query is reached, stating to PostgreSQL that no longer exist tuples available. XDB stores relationships between tables and indexes in main memory, which will affect their performance if there are too many tables of data.

3.2 Experimental Setup

To enable the public sharing of the results, we use the modified queries in [22] on the data set of the TPC-H benchmark [23]. These queries are presented in Table 1.

Table 1: TPC-H queries modified.

Query number	Query
Q3	SELECT SUM(l_extendedprice * (1 - l_discount)) FROM customer, orders, lineitem WHERE c_mktsegment = 'BUILDING' AND c_custkey = o_custkey AND l_orderkey = o_orderkey;
Q7	SELECT SUM(l_extendedprice * (1 - l_discount)) 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 = 'CHINA';
Q10	SELECT SUM(l_extendedprice * (1 - l_discount)) FROM customer, lineitem, orders, nation WHERE c_custkey = o_custkey AND l_orderkey = o_orderkey AND l_returnflag = 'R' AND c_nationkey = n_nationkey;

There experiments were running on a virtual machine with Ubuntu 16.04. This machine has 4GB of memory and 125GB of disk.

We used a three scale factors of the TPCH-benchmark: 1, 10 and 50 for generating our test data. This results in a database that is approximately 1, 10 and 50GB respectively. The queries in the XDB were defined with the same execution time as returned in the VerdictDB, in order to compare the accuracy of

the results in the same amount of time. Each query was executed five times in PostgreSQL, MySQL, VerdictDB and XDB. For each one of them have been the result of each query and the time each took to return a result. Then, with these values were calculated the speedup and the error rate.

3.3 Discussion of results

MySQL was the database engine chosen to assist the VerdictDB in AQP, because the data were stored in MySQL. The speedup ratio is a number that measures the relative performance gain of two systems processing the same problem. In our case it is the improvement in speed of query response time on VerdictDB compared to MySQL and is calculated as follows:

$$\text{Speedup} = \frac{\text{MySQL query response time}}{\text{VerdictDB query response time}} \quad (1)$$

Table 2 presents the speedup of the VerdictDB compared to MySQL. The VerdictDB was nearly 76x faster than MySQL in executing the query Q7 with 50 GB of data. This result is related to the sampling of the VerdictDB. The query Q7 is the query that joins more tables, and all are sampled. So, after the query rewriting with these tables sampled, the response is returned in a short period of time.

Table 2: VerdictDB speedup compared to MySQL.

Query number	Size of TPC-H data		
	1GB	10GB	50GB
Q3	1.44	16.42	17.87
Q7	2.84	43.29	76.13
Q10	2.60	22.85	29.08

According to the results presented in Figure 1, you can see that the speedup increases with the increase of the quantity of data. The speedup also increases when the query has several larger joins, as we can see through the queries Q7 and Q10. This is because MySQL uses a simple multi-join method that looks for rows that match a given column. As such, your disk I / O cost is higher than in VerdictDB, which only uses a sample of the data.

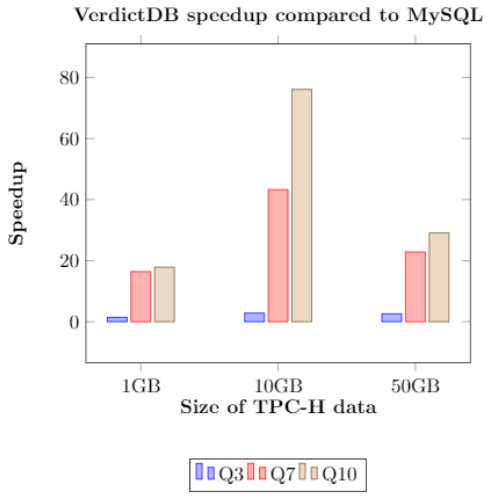

Figure 1: Speedup in VerdictDB compared to MySQL.

The response returned by the VerdictDB was used in the parameter "WITHTIME" of XDB, being also used a confidence interval of 95%. Figure 2 shows the error rate of both tools for Q3, with the different sizes of TPC-H data. According to this figure, the VerdictDB presents higher error rates than the XDB for large volumes of data. The query Q3 is very simple and only involves the join between three tables, but the difference in results for 1 GB of data can be related to memory issues at the time of execution of the query.

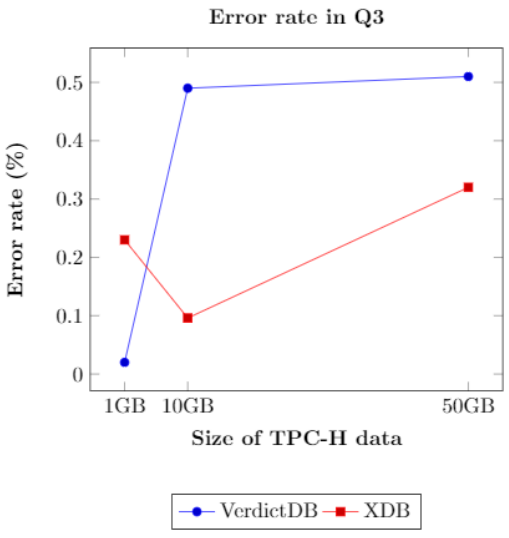

Figure 2: Error rate in Q3.

Figure 3 shows the error rate of both tools for Q7, with the different sizes of TPC-H data. This is the query more complex, involving six joins. The error rate greatly increases compared to the query Q3, because information is required a larger number of tables. The high error rate of VerdictDB in relation to the XDB can be may be related to the fact that VerdictDB has a middleware architecture, all of which calculations about the sampling data are based on SQL.

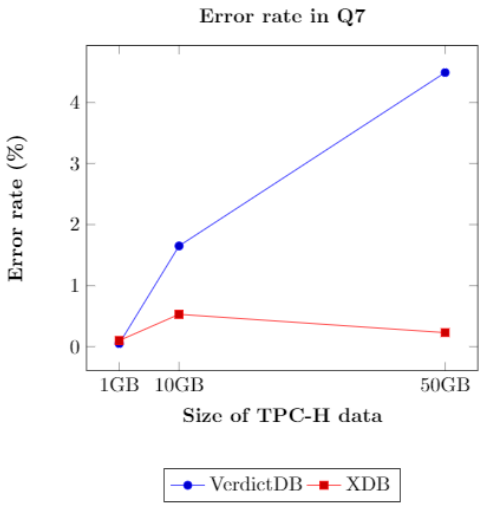

Figure 3: Error rate in Q7.

Figure 4 shows the error rate of both tools for Q10, with the different sizes of TPC-H data. The error rate is still very low in both tools, because there is a very small amount of data to be sampled and are used a few tables. With 50 GB of data error rate is very low in XDB, since this only makes the sampling of a table with selection predicates. VerdictDB uses variational subsampling, in which case only one table is sampled, where the rows of the table belong to several subsamples. This causes a repetition of the data in the samples, increasing the error of the response to the query. Because XDB has a walking plan optimizer, it is possible that the best path has always been selected, since error rates are greatly reduced.

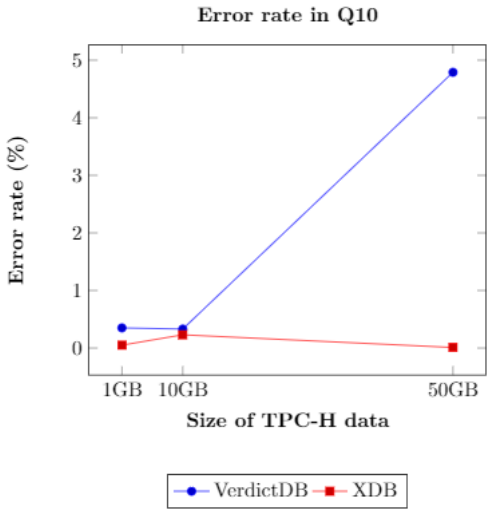

Figure 3: Error rate in Q10.

The query Q3 is the one that presents a lower rate of error, because it uses fewer sample tables. On the other hand, the queries Q7 and Q10 use more sample tables, returning results less accurate.
The XDB give more accurate results than VerdictDB, especially when using many tables in the same query. The variational sampling of VerdictDB may be at the origin of these results, since this is based on randomness. Thus, it is not possible to be sure that a row in a table is not sampled again. If repeated lines are sampled, the sample does not become representative of the population and causes high error rates.

4 CONCLUSIONS AND FUTURE WORK

In this paper, we compare the VerdictDB and the XDB with different scale factors of the TPC-H benchmark in relation to speedup and error rate. Also compared the speedup of VerdictDB compared to MySQL.
We conclude that the VerdictDB can be 76x even faster than MySQL. The VerdictDB and the XDB are good tools for large volumes of data, but the XDB stood out. With the same response time, the XDB returns results with double accuracy.
For small amounts of data, such as 1 GB, it is not justified to use an AQP system. Although we have a speedup of around 20 x greater in AQP, we also have an associated error rate. For this case, if we want a result less fast, but more precise, we use the queries in a traditional DBMS.

As future work, we propose to evaluate more AQP systems in Big Data and cloud environments. We also intend to suggest new algorithms to improve performance and accuracy.

REFERENCES

[1] Mozafari, B. 2017. Approximate Query Engines. Proceedings of the 2017 ACM International Conference on Management of Data - SIGMOD '17.

[2] Han, X., Wang, B., Li, J. and Gao, H. 2017. Efficiently processing deterministic approximate aggregation query on massive data. Knowledge and Information Systems, 57(2), pp.437-473.

[3] Moritz, D. and Fisher, D. 2017. What Users Don't Expect about Exploratory Data Analysis on Approximate Query Processing Systems. Proceedings of the 2nd Workshop on Human-In-the-Loop Data Analytics - HILDA'17.

[4] Sassi, M., Tlili, O. and Ounelli, H. 2012. Approximate Query Processing for Database Flexible Querying with Aggregates. Transactions on Large-Scale Data- and Knowledge-Centered Systems V, pp.1-27.

[5] Moritz,. and Fisher, D. 2017. What Users Don't Expect about Exploratory Data Analysis on Approximate Query Processing Systems. Proceedings of the 2nd Workshop on Human-In-the-Loop Data Analytics - HILDA'17.

[6] Li, K. and Li, G. 2018. Approximate Query Processing: What is New and Where to Go?. Data Science and Engineering.

[7] Inoue, T., Krishna, A. and P. Gopalan, R. 2016. Approximate Query Processing on High Dimensionality Database Tables Using Multidimensional Cluster Sampling View. Journal of Software, 11(1), pp.80-93.

[8] Kraska, T. 2017. Approximate Query Processing for Interactive Data Science. Proceedings of the 2017 ACM International Conference on Management of Data - SIGMOD '17.

[9] Özsu, M. and Liu, L. 2009. Encyclopedia of database systems. New York: Springer.

[10] Vrbsky, S. 1996. A data model for approximate query processing of real-time databases. Data & Knowledge Engineering, 21(1), pp.79-102.

[11] Potti, N. and Patel, J. 2015. DAQ: A New Paradigm for Approximate Query Processing. Proceedings of the VLDB Endowment, 8(9), pp.898-909.

[12] Haas, P. and Hellerstein, J. 1999. Ripple joins for online aggregation. Proceedings of the 1999 ACM SIGMOD international conference on Management of data - SIGMOD '99.

[13] Li, F., Wu, B., Yi, K. and Zhao, Z. 2016. Wander Join. Proceedings of the 2016 International Conference on Management of Data - SIGMOD '16.

[14] Li, F., Wu, B., Yi, K. and Zhao, Z. 2017. Wander Join and XDB. ACM SIGMOD Record, 46(1), pp.33-40.

[15] Agarwal, S., Mozafari, B., Panda, A., Milner, H., Madden, S. and Stoica, I. 2013. BlinkDB. Proceedings of the 8th ACM European Conference on Computer Systems - EuroSys '13.

[16] Park, Y., Mozafari, B., Sorenson, J. and Wang, J. 2018. VerdictDB. Proceedings of the 2018 International Conference on Management of Data - SIGMOD '18.

[17] Goiri, I., Bianchini, R., Nagarakatte, S. and Nguyen, T. 2015. ApproxHadoop. Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS '15.

[18] Chakrabarti, K., Garofalakis, M., Rastogi, R. et al.: Approximate query processing using wavelets. The VLDB Journal 10(2-3), 199-223 (2001)

[19] github.com/mozafari/verdictdb/

[20] github.com/InitialDLab/XDB

[21] docs.verdictdb.org/

[22]github.com/InitialDLab/XDB/tree/master/queries

[23] www.tpc.org/tpch/

Anexo D

Artigo Científico

Este anexo apresenta o artigo a submeter a uma revista.

MDPI

Article

JDBCApprox: an approximate processing system for in-memory queries

Solange Paz [1], Bruno Cabral [1,2] and Jorge Bernardino [2,3,*]

[1] Department of Informatics Engineering, University of Coimbra, Coimbra 3030-290, Portugal; spaz@student.dei.uc.pt; bcabral@dei.uc.pt
[2] Centre of Informatics and Systems of University of Coimbra (CISUC), Coimbra 3030-290, Portugal
[3] ISEC—Coimbra Institute of Engineering, Polytechnic of Coimbra, Coimbra 3030-199, Portugal
* Correspondence: jorge@isec.pt

Version September 5, 2019 submitted to Journal Not Specified

Abstract: To improve the performance of current technologies for data analysis, the approximate processing systems of research have emerged, that ensure the rapid processing of large amounts of data, abdicating from 100% accuracy in response but promoting shorter response times, using only a portion of the data set. Over the last few decades, a large number of techniques for processing approximate research, however these have limitations. The objective of this paper is to propose a new technique of approximate processing of research that mitigated the following shortcomings of current approaches: it does not require any changes to be made to the database since it has a middleware architecture; allows the parameterization of the degree of confidence and the maximum error admitted to the response of a survey and deals with most types of queries. This technique consists of the implementation of a Java library that uses a simple random sampling without repetition to create samples of the database tables and, then use a database with an in-memory configuration to get an acceleration in the response time of the queries. The deployed library can be up to 24 times faster than PostgreSQL and returns in most cases more accurate answers than VerdictDB, this being the system that presents the best results of the state of the art.

Keywords: Approximate query processing; error rate; query response time; random sampling.

1. Introduction

The new strategies for collecting and managing information have generated large amounts of data in companies. This data relates to your customers and their daily business activities [8]. Traditional query processing systems focus solely on providing accurate query responses, with high response times for large amounts of data. In large data storage and writing environments, providing an accurate answer to a complex query can take hours due to the amount of resources required [1]. There are several scenarios where an exact answer may not be required and the user may prefer an approximate answer as it will be faster. First, it is possible to make perfect decisions in many cases without getting an exact answer. This is because query results are useful for making better decisions, and if estimates of an approximate answer lead to the same decisions, it is unnecessary to use accurate values [14]. For example, during polling, as these values will only be used once. Another example is when the query has numeric responses and the total accuracy of the exact answer is not required, for example, a numeric result where only the first precision digits are of interest (such as the first digits of a total in millions). Essentially for these scenarios it has been proposed the approximate response processing that calculates approximate responses very efficiently meeting the performance requirements. Over the past few decades, a large number of approximate query processing techniques have been proposed [7] , ensuring rapid processing of large amounts of data, abdicating 100% accuracy in response but

promoting shorter response times. Processing is performed using only one sample of the data set, which ensures that the result obtained from it is similar to that obtained using the original data set. Although the number of techniques created has been increasing, approximate query processing systems have similar problems in their operation [12]. Most approximate query processing systems require some modification to existing database engines. Another problem is related to incompatibility with Business Intelligence tools, as error estimates returned by approximate query processing systems are not compatible with them. The last problem concerns user interface design as they find it difficult to interpret the true approximation error. In this paper, we introduce a general system architecture for approximate query processing that is based on a technique that we call JDBCApprox. The basic idea is to construct during the pre-processing phase samples based on random sampling without repetition and then, for each query that during the runtime phase, use these samples to provide approximate answers. To increase system performance so that responses are obtained in the shortest possible time, an in-memory configuration database is used. This type of database is fast as it uses data stored in memory instead of data on disk, with faster input/output operations. In addition to the architecture of a query processing system using simple random sampling without repetition, a significant contribution of this paper is the development of two interfaces: GUI and API that support this architecture. These are useful for the developer who wants to use this system in different contexts and for the user who gets the approximate results clearly. Our experimental comparisons demonstrate that JDBCApprox outperforms previously known approximate query processing techniques. The organization of the remainder of the paper is as follows: first, in Section 2, we provide an overview of previous work in approximate query processing. Then, in Section 3, we describe the JDBCApprox architecture in detail. In Section 4 we give experimental evaluation of JDBCApprox. Finally, Section 5 states the conclusions and proposes some future work.

2. Related Work

In the last years, several works have appeared in the area of approximate query processing based on sampling [19], [3], [5], [16], [10]. The sampling technique of these systems differs. For example, BlinkDB [3] creates samples based on column sets and AQUA [19] creates a single stratified sample. To support join operation in aproximate query processing systems, Hass and Hellerstein created a new technique called ripple joins [10]. In Ripple joins use large amounts of memory during the join process, and to solve this problem has been created the ripple join hash [13]. However, both techniques have limitations on the queries they support. A new biased sample-based technique has been created where rows are selected based on the results of previous queries [20]. This technique is called SciBORQ and targets exploratory scientific analysis, producing quality responses and with error limits in pre-defined time frames. Several approximation techniques have been proposed using also sketches [7] and online aggregation [17], [11]. The sketches were used in an approximate aggregation technique for sensor database [6]. In this system aggregate query responses such as SUM and COUNT are approximate, producing accurate results with a low computation overhead. The wavelets are also used in approximate query processing systems. In [4] the authors used the wavelets and later compared them with sampling techniques and histograms. They concluded that histograms perform worse than wavelets. Excellent work has been done on using bootstrap over apriori sample, where bootstrap is used to determine a priori which sample is suitable for a given query [2]. In this paper, the authors show that it is possible to quickly and accurately diagnose errors in error estimation for a query. They also show that it is possible to implement a system that produces approximate responses and reliable error bars at high speeds. Another interesting technique is the use of mapreduce tools to approximate query processing, an example of this is ApproxHadoop [9]. This system creating and running approximation-enable MapReduce programs.ApproxHadoop can reduce execution times by up to 32x with a 1% error and a 95% confidence interval. So far there is only one rough query processing system that has a middleware architecture, this is VerdictDB [18]. This system exploits a new technique for error estimation called variational subsamplin, incurring less than 2.6% relative

error and providing a speed increase of up to 171x compared to existing engines like Impala, Spark SQL and Amazon RedShift.

3. System Overview

In this section, we overview the system we have built based on random sampling without repetition. Section 3.1 explains JDBCApprox's architecture. Section 3.2 explicates supported SQL query types. Section 3.3 describes the sampling technique used in JDBCApprox. Lastly, Section 3.4 discusses JDBCApprox's current limitations.

3.1. Architecture

Figure 1 illustrates the architecture of JDBCApprox.

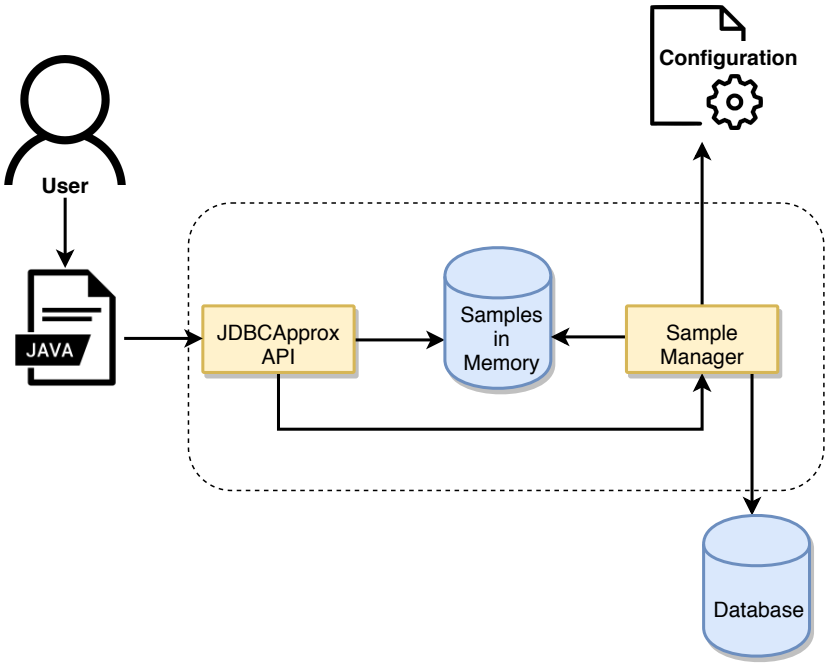

Figure 1. JDBCApprox Architecture.

JDBCApprox is accessed through Java code, which uses its API. It has the function of dealing with the queries, through its division and rewriting, sending to the sample manager the name of the tables that the query has. The sample manager then checks to see if the samples exist in memory and if there is a rewritten search is performed. Otherwise, the sample manager accesses the information in the configuration file to create samples and copy them to memory.
Initially, when the user types a search, it is split. This division consists of separating the search through its clauses (SELECT, FROM, WHERE, GROUP BY, ORDER BY) and extracting the names from tables present in the FROM clause. We also identify the aggregation functions used and their position in the search, for later rewriting. The table names are then sent to the sample manager, who checks whether samples of these tables are present in the in-memory configuration database. After checking the existence of the samples, the metadata of the tables that are present in PostgreSQL is consulted, in order to identify which table has the largest number of foreign keys and which foreign keys of the remaining tables. The configuration file is then accessed to determine the size of the sample according to the equation 2 by indicating the degree of confidence and the maximum error percentage to be admitted. When the sample size is determined, it is created randomly without repetition. The remaining samples are then created according to the following rule: If the table to be sampled has only one foreign key, the sample is created using a join with the table that has that foreign key.Once samples are created, they must be copied to memory to speed up response to searches. Thus, the

PostgreSQL samples are copied to a memory configuration database using a solution called pg2sqlite [**?**]. After copying the samples to memory, the search needs to be rewritten so that when it is run it will produce an estimated result from the samples. To estimate the result of a search, the result of aggregate functions SUM or COUNT is divided by the percentage of the sample, obtained from the equation 1.

$$Sample\ percent = \frac{Largest\ table\ size}{Sample\ size} \tag{1}$$

In the end, after the query has been rewritten, it is executed. This consists of performing the rewritten search in memory. This setting was used for queries to be answered faster because in-memory databases are faster than disk databases. In in-memory databases disk access is slower than memory access and internal optimization algorithms are simpler and execute fewer instructions.

3.2. Suported Queries

JDBCApprox supports aggregate queries with the following conditions:

1. **Aggregates**
   Any number of SUM, COUNT, AVG, MAX, MIN, STDDEV and VARIANCE aggregates can appear in the select clause. These aggregates can also used a derived attribute.
2. **Grouping**
   The group by clauses are supported for both stored and derived attributes.
3. **Joins**
   Foreign-key joins with any number of tables are supported.
4. **Selections**
   Supports equality and inequality comparisons for categorical and numerical attributes , including the IN operator. Supports logical operators such as AND and OR comparisons with subqueries.
5. **Table sources**
   Supported derived tables or base tables joined with a joins. The derived table can be a select statement with or without aggregate functions.
6. **Other clauses**
   Supports group by, order by, limit and having clauses.

3.3. Sampling Technique

In the proposed approach a simple random sampling without repetition was used since it does not select the same element for the sample more than once and is indicated for large data sets to allow more elements to be randomly selected for the sample. In simple random sampling without repetition [15] each table is treated independently and in the end the resulting samples from each table are joined. The final results through this technique yielded large percentages of error in estimating the size of a junction, as it was not guaranteed that the data sample from one table would cover all data from another sample. This sampling technique is beneficial when you want to get a rough answer from a search using only one table. In order to reduce the relative percentage error in queries using multiple tables, a new sampling strategy was designed. The basic idea is to apply a simple random sampling without repetition to the table in the database that has the most foreign keys, and then to select the rows of the remaining tables that join this sample.
The number of data selected from the table that has the most foreign keys is determined by the equation 2.

$$n = \frac{N \times Z^2 \times p \times (1-p)}{(N-1) \times e^2 + Z^2 \times p \times (1-p)} \tag{2}$$

At where:

n = Number of sample data to calculate.
N = Number of rows in the table for which to sample.

Z = Deviation from the mean value to achieve the desired degree of confidence. Depending on the desired degree of confidence, a value that is given by the shape of the Gaussian distribution is used. **e** = Maximum error percentage to be admitted.
p = Proportion of data not belonging to the category to study.
The proportion p appears in the formula, because when a population is very uniform, convergence to a normal population is more accurate and allows you to reduce the sample size. Since it is not possible to identify the proportion of data to sample a table, the following scenario is used: the population is evenly distributed, so $p = 50\%$.
Once the amount of data has been determined, the sample is created by selecting data at random without repetition until it reaches that amount. Then the relationships with the other tables are identified and the samples are created according to the following rule: if the table to be sampled (Table A) has only one foreign key (Table B), the The sample is created using a join with the table that has this foreign key (join between tables A and B). If the table (Table A) has multiple foreign keys, the sample is created using a join with the table that contains the largest number of records and has that foreign key.

3.4. Limitations

JDBCApprox has a simple, non-repeating random sampling engine to roughly respond to queries. A sample of a given table has a small number of rows randomly selected from the original table. This gives approximate answers that are very far from the original answer in a specific case. This is when you want an approximate response to a query that uses data grouped using a unique attribute. For example, for the following query:

```
SELECT
    c_custkey, c_name, sum(l_extendedprice)
FROM
    customer, orders, lineitem
WHERE
    c_custkey = o_custkey
    AND l_orderkey = o_orderkey
GROUP BY
    c_custkey,
    c_name;
```

The attributes *c_custkey* and *c_name* are unique, meaning there is only one row in the *customer* table for each of these attributes. When selecting a sample from the *customer* table, it cannot contain all the data for the attributes present in the query. Thus, JDBCApprox has a limitation for this type of queries.

4. EXPERIMENTAL EVALUATION

Our experiments aim to (i) quantify the number of queries that benefit from JDBCApprox and their average speedup (Section 4.2) and study the query response relative error (Section 4.3). In our experiments we use VerdictDB as a means of comparison with JDBCApprox.

4.1. Experimental Setup

Datasets and Query Workloads - For our experiment, we used the TPC-H dataset described here: this is a analytical benchmark with 22 query types, 21 of which contain at least one aggregate function . We used two scale factors of 1 and 10, i.e., the total data size was 1GB and 10GB respectively. To facilitate the analysis of results, we used TPC-H queries that differed from the others due to the number of joins and number of tables used. Tables that did not run in VerdictDB were also excluded from the experimental evaluation. The queries used and their characteristics are shown in the Table 1.

Table 1. Characterization of used TPC-H queries in terms of SQL.

Queries	Key Characteristics of Queries
Q2	1 group by, 4 join, 5 tables
Q3	1 group by, 2 join, 3 tables
Q4	1 group by, 1 join, 2 tables
Q5	1 group by, 5 join, 6 tables
Q6	0 group by, 0 join, 1 table
Q14	0 group by, 1 join, 2 table

Metrics - We used two metrics: (i) speedup and (ii) relative percentage error. Speedup measures the relative performance of two systems that process the same problem, that is, the improvement in the speed of execution of a task performed on two similar architectures with similar features. In the experimental evaluation the speedup of VerdictDB was calculated through the equation 3 and JDBCApprox in relation to PostgreSQL using the equation 4, where time refers to execution time.

$$VerdictDB\ \ Speedup = \frac{PostgreSQL\ \ Time}{VerdictDB\ \ Time} \tag{3}$$

$$JDBCApprox\ \ Speedup = \frac{PostgreSQL\ \ Time}{JDBCApprox\ \ Time} \tag{4}$$

Speedup values allowed comparing the difference between VerdictDB and JDBCApprox gain.
The relative percentage error (E) allows to identify the percentage error of an approximate value relative to an exact value. In the experimental evaluation we calculated the relative error percentage of the approximate search response returned by VerdictDB and JDBCApprox, in relation to the exact response returned by PostgreSQL. The relative percentage error of a query allowed to evaluate the accuracy of the responses of the two approaches analyzed and was calculated through the equation 5, where $y = exact\ \ answer$ and $y' = approximate\ \ answer$.

$$E = \left| \frac{y - y'}{y} \right| \times 100 \tag{5}$$

Thirty executions were performed for each search, and whenever a new search was performed the PostgreSQL service was restarted so that all searches were performed under the same conditions. The first run on PostgreSQL was excluded from analysis, as PostgreSQL loads data from disk to memory, which is a slower process. For each query, after thirty runs, the result and response time averaged.

4.2. Query Speedup

The graph in Figure 3 illustrates the speedup for queries performed on JDBCApprox and VerdictDB for 1GB of data.

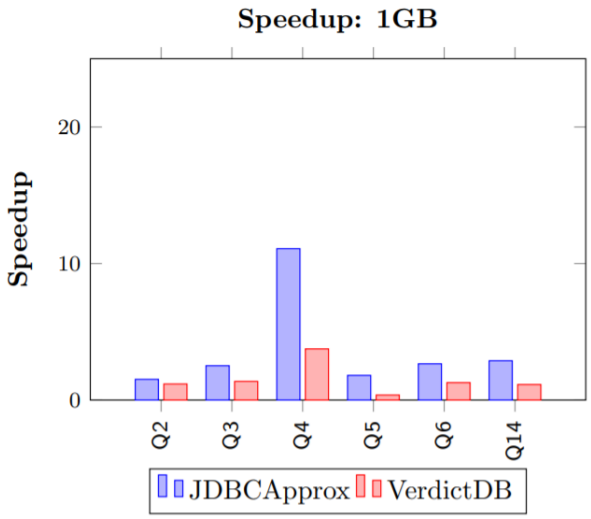

Figure 2. Speedup for each query with 1GB of data.

Running the Q5 query through VerdictDB does not get any speedup since they use six tables and a large number of constraints in the where clause. With the high complexity of this query, its execution allocated a lot of RAM, being slower than PostgreSQL. The speedup of about 11x in Q4 is related to the fact that the orders table sample is directly created from the lineitem table sample, and there is no data other than those joining these two. In addition, PostgreSQL has longer runtimes with correlated subqueries. The main reason why speedup with JDBCApprox is superior is to use an in-memory database. This type of database configuration performs better because reading and writing data to memory is faster than performing the same disk operations.
The graph in Figure **??** illustrates the speedup for queries performed on JDBCApprox and VerdictDB for 1GB of data.

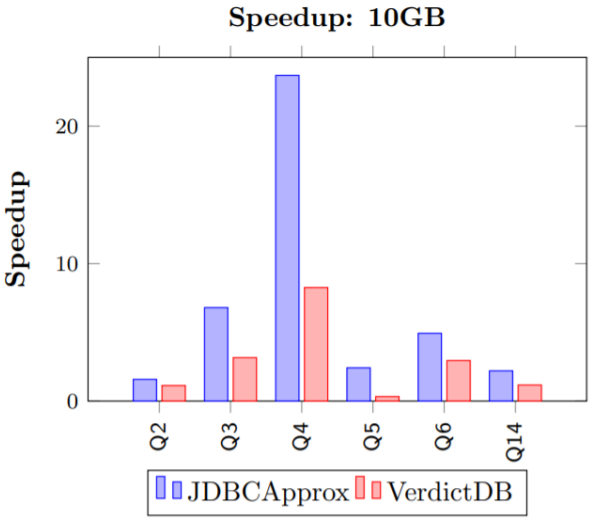

Figure 3. Speedup for each query with 10GB of data.

4.3. Query Response Relative Error

Next, the results obtained in relation to the relative percentage error in the result of each search will be presented.
Table 2 shows the percent relative error recorded in Q2 lookup using VerdictDB and JDBCApprox for 1GB and 10GB. In general, the percent relative error decreases as the amount of data increases as the selected sample will have more database records. When Q2 lookup is performed on VerdictDB and later on JDBCApprox, the percentage relative error recorded is mostly higher on VerdictDB. The

different error values are related to how VerdictDB uses data to respond to the search as it divides the data by block and executes it block by block until the values converge and the error is less than a set value. The relative percentage error for the RUSSIA group is the highest, as this is the group with the lowest number of survey elements.

Table 2. Percent relative error(%) in query Q2 for each group.

	Group	JDBCApprox	VerdictDB
1 GB	FRANCE	0.69	1.14
	GERMANY	0.64	0.95
	ROMANIA	1.68	1.45
	RUSSIA	1.81	2.10
	UK	0.54	0.41
10 GB	FRANCE	0.45	1.10
	GERMANY	0.18	0.05
	ROMANIA	0.41	0.25
	RUSSIA	1	1.54
	UK	0.02	0.61

Table 3 shows the percent relative error recorded in Q3 lookup using VerdictDB and JDBCApprox for 1GB and 10GB. Query Q3 has about the same number of records in each group, so the relative error is not influenced by this factor. This error value is higher when using 10GB of data, as the values selected for the sample might not have been the ones that would best represent the original table. In addition, the sum of two values is estimated, with the highest relative error.

Table 3. Percent relative error(%) in query Q3 for each group.

	Group	JDBCApprox	VerdictDB
1 GB	AUTOMOBILE	0.59	0.27
	BUILDING	0.47	0.9
10 GB	AUTOMOBILE	1.69	2.89
	BUILDING	2.09	3.88

Table 4 shows the percent relative error recorded in Q4 lookup using VerdictDB and JDBCApprox for 1GB and 10GB. Query Q4 uses a sub-query that selects all rowitem table records according to two constraints, which will later influence the relative error. That is, because the lineitem table is a sample of the original table, it will not have all the records required for the main search.

Table 4. Percent relative error(%) in query Q4 for each group.

	Group	JDBCApprox	VerdictDB
1 GB	1-URGENT	10.01	7.3
	2-HIGH	9.49	12.17
	3-MEDIUM	6.70	8.71
	4-NOT SPECIFIED	8.98	9.93
	5-SLOW	7.27	9.08
10 GB	1-URGENT	8.65	9.3
	2-HIGH	8.31	11.17
	3-MEDIUM	7.83	9.84
	4-NOT SPECIFIED	8.13	8.70
	5-SLOW	8.29	7.52

Table 5 shows the percent relative error recorded in Q5 lookup using VerdictDB and JDBCApprox for 1GB and 10GB. Query Q5 uses six tables and many constraints in the WHERE clause that influence relative error. This reaches a maximum value of 3 % for JDBCApprox and 5 % for VerdictDB. The JAPAN group has fewer records, registering a higher relative error with 1GB of data.

Table 5. Percent relative error(%) in query Q5 for each group.

	Group	JDBCApprox	VerdictDB
1 GB	CHINA	2.70	2.63
	INDIA	0.26	0.83
	INDONESIA	0.60	1.07
	JAPAN	3.13	2.75
	VIETNAM	2.11	1.12
10 GB	CHINA	0.78	3.63
	INDIA	0.32	1
	INDONESIA	1.38	2.61
	JAPAN	0.52	1.51
	VIETNAM	1.42	5

Table 6 shows the percent relative error recorded in Q6 lookup using VerdictDB and JDBCApprox for 1GB and 10GB. Although the relative percentage error using VerdictDB is different from that obtained when using JDBCApprox, this value is not significant. When Q6 search is performed with only 1GB of data there is a difference of about 0.4%, in contrast this difference decreases to about 0.15% when 10GB of data is used. This difference is related to the randomness of the data selected to answer the survey.

Table 6. Percent relative error(%) in query Q6 for each group.

	JDBCApprox	VerdictDB
1 GB	0.52	0.10
10 GB	0.04	0.22

Table 7 shows the percent relative error recorded in Q14 lookup using VerdictDB and JDBCApprox for 1GB and 10GB. Query Q14 uses only two tables and has few restrictions on the WHERE clause, which results in lower relative error values.

Table 7. Percent relative error(%) in query Q14 for each group.

	JDBCApprox	**VerdictDB**
1 GB	0.31	0.64
10 GB	0.34	1.63

5. Conclusions and future work

By using simple random sampling without repetition to create samples, which were later copied to an in-memory configuration database, it was possible to build a system with a middleware architecture that handles most types of researches. With the possibility of parameterizing the degree of confidence and the maximum error allowed for a survey response, users can understand how accurate an answer

is. In addition, by building a user interface, the user can more easily understand how JDBCApprox works and what responses it returns.
The developed library has more accurate results than existing approaches and can be 24 times faster, and in most cases two to three times faster.
As future work, we propose testing with more data and different data sets to understand how the proposed approach behaves under different conditions. We also propose to use a different technique when using join searches, to achieve better results when the number of records in each group is very different.

References

1. Acharya, S., Gibbons, P. B., Poosala, V., and Ramaswamy,S. (1999). Join synopses for approximate query answering. InSIGMOD 1999, *Proceedings ACM SIGMOD International Conference on Management of Data*, **June 1-3, 1999**, Philadelphia, Pennsylvania,USA., pages 275–286
2. Agarwal, S., Milner, H., Kleiner, A., Talwalkar, A., Jordan,M. I., Madden, S., Mozafari, B., and Stoica, I. (2014).Knowing when you're wrong: building fast and reliable approximate query processing systems. *In International Conference on Management of Data*, SIGMOD 2014, Snowbird, UT, USA, **June 22-27, 2014**,pages 481–492.
3. Agarwal, S., Mozafari, B., Panda, A., Milner, H., Madden, S., and Stoica, I. (2013). Blinkdb: queries withbounded errors and bounded response times on verylarge data. *InEighth Eurosys Conference 2013*, EuroSys '13, Prague, Czech Republic, **April 14-17** , 2013,pages 29–42.
4. Chakrabarti, K., Garofalakis, M. N., Rastogi, R., andShim, K. (2001). Approximate query processing using wavelets.VLDB J., 10(2-3):199–223.
5. Chaudhuri, S., Das, G., and Narasayya, V. (2007). Optimized stratified sampling for approximate query processing.ACM Trans. Database Syst., 32(2).
6. Considine, J., Li, F., Kollios, G., and Byers, J. W.(2004). Approximate aggregation techniques for sensor databases. *In Proceedings of the 20th International Conference on Data Engineering* , ICDE 2004, **30 March - 2 April 2004**, Boston, MA, USA, pages449–460.
7. Cormode, G., Garofalakis, M. N., Haas, P. J., and Jermaine,C. (2012). Synopses for massive data: Samples, histograms, wavelets, sketches.Foundations and Trendsin Databases, 4(1-3):1–294.
8. Ganti, V., Lee, M., and Ramakrishnan, R. (2000). ICICLES: selftuning samples for approximate query answering. InVLDB 2000, *Proceedings of 26th International Conference on Very Large Data Bases*, **September 10-14, 2000**, Cairo, Egypt, pages 176–187.
9. Goiri, I., Bianchini, R., Nagarakatte, S., and Nguyen, T. D.(2015). Approxhadoop: Bringing approximations tomapreduce frameworks. *In Proceedings of the Twentieth International Conference on Architectural Supportfor Programming Languages and Operating Systems*,ASPLOS '15, Istanbul, Turkey, **March 14-18, 2015**,pages 383–397.
10. Haas, P. J. and Hellerstein, J. M. (1999). Ripple joins for online aggregation. *InSIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data* , **June 1-3, 1999**, Philadelphia, Pennsylvania, USA., pages 287–298.
11. Li, F., Wu, B., Yi, K., and Zhao, Z. (2019). Wander join and xdb: Online aggregation via random walks. ACMTrans. Database Syst., 44(1):2:1–2:41.
12. Li, K. and Li, G. (2018). Approximate query processing: What is new and where to go? A survey on approximate query processing. Data Science and Engineering, 3(4):379–397.
13. Luo, G., Ellmann, C. J., Haas, P. J., and Naughton, J. F.(2002). A scalable hash ripple join algorithm. *In Proceedings of the 2002 ACM SIGMOD InternationalConference on Management of Data, Madison*, Wisconsin, USA, **June 3-6, 2002**, pages 252–262.
14. Mozafari, B. (2017). Approximate query engines: Commercial challenges and research opportunities. *In Proceedings of the 2017 ACM International Conferenceon Management of Data*, SIGMOD Conference 2017,Chicago, IL, USA, **May 14-19**, 2017, pages 521–524.
15. Olken, F. and Rotem, D. (1994). Random sampling from databases a survey. Statistics and Computing, 5:25–42.

16. Olston, C., Bortnikov, E., Elmeleegy, K., Junqueira, F.,and Reed, B. (2009). Interactive analysis of webscale data. *In CIDR 2009, Fourth Biennial Conferenceon Innovative Data Systems Research*, Asilomar, CA,USA, **January 4-7, 2009**, Online Proceedings.
17. Pansare, N., Borkar, V. R., Jermaine, C., and Condie, T.(2011). Online aggregation for large mapreduce jobs.*PVLDB*, 4(11):1135–1145.
18. Park, Y., Mozafari, B., Sorenson, J., and Wang, J. (2018).Verdictdb: Universalizing approximate query processing. *In Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018*, Houston, TX, USA, **June 10-15, 2018** , pages1461–1476.
19. Sellis, T. K. (2000). Review the aqua approximate query answering system.ACM SIGMOD Digital Review, 2.Sidirourgos, L., Kersten, M., and Boncz, P. (2011a). Sciborq: Scientific data management with bounds on run-time and quality. *In In Proc. of the Intl Conf. on Inno- vative Data Systems Research* (CIDR, pages 296–301.
20. Sidirourgos, L., Kersten, M. L., and Boncz, P. A. (2011b).Sciborq: Scientific data management with bounds onruntime and quality. *In CIDR 2011, Fifth Biennial Conference on Innovative Data Systems Research ,Asilomar* , CA, USA, **January 9-12, 2011**, Online Proceedings, pages 296–301.