Fulano de Tal

Título da Monografia

Trabalho de conclusão de curso apresentada ao Departamento de Matemática, Estatística e Computação (DMEC), da Universidade Estadual Paulista "Júlio de Mesquita Filho", sob o título **"Título da Monografia"**.

Orientador: Prof. Dr. Nome do Orientador

Co-orientador: Prof. Dr. Nome do Coorientador

Presidente Prudente

2010

Agradecimentos

A realização do presente curso foi possível devido à colaboração de muitas pessoas que me auxiliaram durante os 5 anos de curso. Manifesto assim minha gratidão:

primeiramente a Deus, que sempre me ajudou a não desistir dessa longa caminhada, e que sempre me acompanha durante a execução de minhas tarefas.

outros agradecimentos .....

Resumo

Elemento obrigatório, constituído de uma sequência de frases concisas e objetivas e não de uma simples enumeração de tópicos. Deve conter de 150 a 500 palavras, seguido logo abaixo das palavras-chave (conforme NBR 6028).

Elemento obrigatório. Consiste em uma versão do resumo em idioma de divulgação internacional (em inglês Abstract, em espanhol Resumen, em francês Résumé). Também deve ser seguido de palavras-chave,

palavras-chave: palavras representativas do conteúdo do trabalho, separadas entre si por ponto (.) e finalizadas também por ponto. Devem aparecer logo abaixo do resumo, antecedidas da expressão “palavras-chave:”.

Abstract

Elemento obrigatório. Consiste em uma versão do resumo em idioma de divulgação internacional (em inglês Abstract, em espanhol Resumen, em francês Résumé). Também deve ser seguido de palavras-chave, na língua.

keywords: palavras representativas do conteúdo do trabalho, separadas entre si por ponto (.) e finalizadas também por ponto. Devem aparecer logo abaixo do resumo, antecedidas da expressão “keywords:”.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Bancos de Dados Não Relacionais - NoSQL

Em um trabalho acadêmico intitulado "Selected Topics on Software-Technology Ultra-Large Scale Sites"(Tópicos Selecionados na Tecnologia de Softwares de Sites de Escala Ultra Grande) (STRAUCH; SITES; KRIHA, 2011) na Stuttgart Media University(Universidade de Mídia de Stuttgart) Christof Strauch reúne várias referências e tópicos sobre todo o assunto de base do NoSQL, conforme seu trabalho, os bancos de dados relacionais são os tipos predominantes para armazenamento de dados estruturados, desde 1970 este tipo de base de dados se suporta em uma aritmética relacional com o proliferado uso do SQL(Secure Query Language) para inserir, alterar, recuperar e remover os dados. Bases de dados não convencionais eram utilizadas apenas para aplicações de algum nicho específico como projetos utilizando SOAP que utiliza o padrão XML, porém o pensamento por cima desse uso disseminado "um tamanho para caber todos"tem sido questionado tanto pelas empresas como pelos acadêmicos devido ao aumento de usuários e volume de dados no meio.

Conforme segue na argumentação de Christof Strauch, para suprir essa demanda algumas alternativas começam a ser propostas, vários outros modelos são criados para abrangir mais nichos antes dominados pelos SGBDRs, esses modeloss adotam caminhos não convencionais nem sempre garantindo a ACID (Atomicidade, Consistência, Isolamento e Durabilidade) dos dados, porém conseguem optimizar o espaço de armazenamento e a performance das bases de dados, não necessariamente se desfazendo do uso do SQL, por isso a sigla NoSQL significa "*not only secure query language*".

1.1 Definição

Existem dois pontos de vista não equivalentes sobre o princípio do NoSQL, o NoSQL Archive (ARCHIVE, 2009) apresenta uma visão seguindo o termo "NonSQL"que significa uma base de dados sem SQL em tudo, no artigo "NoSQL Databases"(Base de Dados NoSQL) (WEBER, 2010) Silvan Weber define outro ponto de vista mais aceito sendo "Not only SQL"(Não apenas SQL), ou seja, base de dados que não é baseada em SQL porém, pode ter ou não funcionalidades SQL. Além do conceito presente no nome, Weber levanta outras características fundamentais:

- Armazenamento Distribuido: Um dos maiores intuítos e motivador para o desenvolvimento de bases NoSQL é a necessidade de distribuir a carga de armazenamento e acessos das bases a vários servidores porém, mesmo dividido em vários servidores, a base continua interagindo com os outros sistemas como uma.
- Fonte aberto: Esse princípio assegura que o código fonte de bases NoSQL estejam disponíveis para que possam ser alterados para melhor se adaptar a necessidade de cada aplicação, diferentemente de SGBDRs comerciais que, além de serem muito onerosos, nem sempre atendem aos requisitos das aplicações
- Escalável horizontalmente: Uma das necessidades de ter um armazenamento distribuído é para que quanto mais servidores ligados à base maior será a performance da base por completo, seguindo uma linha de proporção próxima do linear.

1.2 Aspectos Comuns ao NoSQL

Esta seção destina-se a criar uma base sólida sobre os principais aspectos e conceitos que envolvem o NoSQL, além de apontar diferenças e temas relacionados com SGBDs também serão abordados teoremas e tópicos específicos sobre o NoSQL

1.2.1 Recapitulando SGBDRs

Em uma tese de mestrado intitulada "Extracting Data from NoSQL Databases. A Step towards Interactive Visual Analysis of NoSQL Data"(Extração de dados de bases NoSQL. Um passo avante para análise de visualização interativa de NoSQL) (NÄSHOLM, 2012) da Chalmers University of Technology(Chalmers, Universidade de Tecnologia) Petter Näsholm apresenta uma recapitulação rápida sobre as bases de dados convencionais.

SGBDRs se baseiam no conceito de tabelas com conjunto de atributos podendo ser chamados de colunas da tabela, cada atributo é associado a um tipo de dado(como inteiro, real ou string). Cada linha desta tabela é considerada uma tupla, ao passo que os atributos dentro de cada tupla podem ser preenchidos com componentes referente ao seu tipo. As tabelas também podem ter colunas com informações sobre a relação entre elas mesmas.

1.2.2 ACID e BASE

Petter Näsholm também aborda as garantias que a maioria dos SGBDRs adotam sendo elas a sigla ACID(Atomicidade, Consistência, Isolação, Durabilidade) onde Atomicidade é a garantia de que ou uma trasação é executada de forma completa ou não é executada de forma alguma, a Consistência garante que toda transação saia de um estado da base válido para outro, Isolação é a garantia de que não interfere outra no tempo em que estão ocorrendo e a Durabilidade garante que o efeito de uma transação deve persistir e, portanto, nunca ser perdido. No trabalho de Christof Strauch o autor apresenta o BASE que são as garantias dos NoSQL:

Basic available(Basicamente disponível): Uma aplicação funciona basicamente todo o tempo.

Soft-state(Estado "leve, macio", simples): Não precisa ser consistente o tempo todo.

Eventual consistency(Eventual consistência): Em algum momento conhecido haverá consistência.

O aspecto com maior diferença entre os modelos é a parte da consistência, no modelo ACID existe uma consistência estrita enquanto no base há uma consistência eventual. A consistência estrita implica que toda operação de leitura deve retornar resultados a partir da última escrita, devido a isso as operações de leitura e escrita devem ser feitas no mesmo servidor ou obedecer a um protocolo de transação distribuída, porém, como veremos abaixo pelo teorema CAP, essa forma de operar implica ou na perda da disponibilidade dos dados ou na tolerância do particionamento dos mesmos. A consistência eventual permite que leitores acessem a base conforme os dados vão sendo escritos, não importando necessariamente se aquele registro foi o último a ser escrito, eventualmente em um estado estável será retornado o último registro escrito, até isso ocorrer o sistema pode estar em um estado de inconsistência.

Além das garantias colocadas acima Christof Strauch lista outras que também podem ser fornecidas:

RYOW - Read Your Own Writes(leia seus próprios escritos): O leitor deve visualizar imediatamente aquilo que ele próprio escreveu, independentemente se o leitor acessa pelo próprio servidor que escreveu ou por outro.

Consistência da Sessão: Acompanha a garantia do RYOW porém apenas para a sessão que o leitor estiver conectado, ou seja, caso escreva os dados por um servidor e tenter ler por outro a consistência não será garantida.

Consistência ocasional: se foi escrito y após ser lido x, qualquer leitor que ver y também verá x.

1.2.3 Teorema CAP

O Teorema CAP que, conforme Christof Strauch, é muito adotado pela comunidade NoSQL, apresenta um conceito referente as limitações estritamente associadas as capacidades que oferecem.

A sigla CAP se refere à Consistência(Consistency), Disponibilidade(Availability), Tolerância ao particionamento(Partition Tolerance), sendo respectivamente a habilidade de manter os dados compartilhados disponíveis a todos os leitores após um escritor executar um update, a habilidade de continuar disponível mesmo com a falta de algum cluster ou parte do sistema ficar indisponível e a habilidade de continuar operando mesmo que um servidor não consiga se conectar a outro servidor da mesma aplicação também entendido como a habilidade de poder adicionar ou remover nós de uma rede particionada. Este Teorema propõe que só é possível para qualquer aplicação combinar duas dentre as três características, ou seja ela pode ter muita consistencia e disponibilidade porém com pouca tolerância ao particionamento, muita consistencia e tolerância ao particionamento porém com pouca disponibilidade ou muita disponibilidade e tolerância ao particionamento e pouca consistência.

Observando pelo teorema CAP os NoSQL buscam ofertar alternativas para aplicações que nem sempre dependem de consistência ou disponibilidade ou tolerância ao particionamento.

Também existem outros aspectos mais profundos mas que não seram abordados neste documento como o versionamento do conjunto dos dados, os cenários distribuídos e o particionamento. Os aspectos vistos acima nos dão aparato suficiente para progredir o trabalho focando nas questões principais sobre as necessidades da aplicação e o uso de diferentes tipos de SGBDs para qual se encaixam melhor.

1.3 Formas de Armazenamento

Nesta seção seram abordados algums tipos de bases NoSQL, o modelo orientado à documento utilizando JSON será aprofundado pois foi escolhido para dar suporte a este trabalho devido a sua fácil integração com a ferramenta de visualização de dados escolhida.

1.3.1 Armazenamento baseado em Chave-Valor

Bases de dados Chave-Valor são extremamente simples, existe uma chave única, geralmente gerada por algum algoritmo de Hash, associada a um valor que pode ser de diferentes tipos como: string, inteiros, reais, cadeias de bytes. Algumas bases de dados não implementão nem a verificação de tipo.

Tem a vantagem de ter um modelo de dados simples, feito com foco na rápida e eficiente manipulação dos dados para armazenamento distribuído sendo totalmente escalável. Alguns de seus exemplos: Google¿s BigTable[1], Memcached[2], Redis[3].

```
(customerIds,            [12, 67])
(customerId:12:name,     Smith)
(customerId:12:location, Seattle)
(customerId:67:name,     Andersson)
(customerId:67:location, Stockholm)
(customerId:67:clubCard, 82029)
```

Figura 1: Exemplo de Chave e Valor em Redis (NÄSHOLM, 2012).

Algumas bases são construídas para funcionarem na memória RAM, como é o caso do Memcached, isso permite uma performance muito superior porém implica em um limite de espaço reduzido e o risco de perda de dados no caso de quedas de energia.

1.3.2 Armazenamento orientado a Documento

Este tipo de armazenamento pode ser entendido como um caso particular de uma base Chave-Valor, a diferença é que a base deve ler e escrever em algum tipo conhecido de arquivo e(ou) estrutura. Utiliza documentos com diferentes tipos de conjuntos de dados, alguns tipos de documento são de fácil interpretação para humanos, dentre eles: XML,

[1]https://cloud.google.com/bigtable/, acessado em 16 de Maio de 2016
[2]https://memcached.org/, acessado em 22 de Maio de 2016
[3]http://redis.io/, acessado em 25 de Maio de 2016

JSON, XAML. Também é possível acessar e alterar diretamente tais documentos pelo servidor.

O esquema que existe nas bases é a estrutura utilizada no próprio arquivo, sem relação nenhuma dentre os documentos da base, cada documento funciona independente do outro, no caso de bases que os arquivos dependem são chamadas de semi-estruturadas.

Como a estrutura dos arquivos é algo arbitrário campos sem dados não precisam ser preenchidos e nenhum espaço de armazenamento precisa ser reservado. Dentre os tipos existentes estão: CouchDB[1] e MongoDB[2]

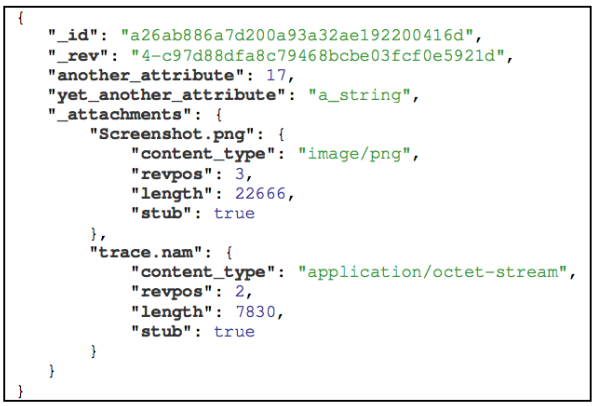

Figura 2: Exemplo de um documento JSON em CouchDB (WEBER, 2010).

1.3.3 Banco de Dados Orientado a Colunas

Diferentemente de bases SQL onde o conjunto de dados é armazenado em uma unidade, este modelo armazena um atributo do conjunto de dados em uma unidade, exemplo: Supondo termos 3 atributos nome, idade, telefone, com o conjunto de dados "Julio", 18, 33821522; "Lucas", 20, 82331248; "Marina", 25, 39031169. Organizando estes dados no formato orientado a colunas eles ficam nesta disposição: "Julio", "Marina", "Lucas"; 18, 20, 25; 33821522, 82331248, 39031169.

O uso dos atributos separados por coluna garante uma rápida análise dos dados mesmo em grande quantidade, pois estão descritos sucessivamente. Então análise como obter a média retornam rapidamente sem percorer outros atributos desnecessários. As bases mais conhecidas são: HBase[3] e Cassandra[4]

[1]http://couchdb.apache.org/, acessado em 13 de Abril de 2016
[2]https://www.mongodb.com/, acessado em 13 de Abril de 2016
[3]https://hbase.apache.org/, acessado em 23 de Junho de 2016
[4]http://cassandra.apache.org/, acessado em 24 de Junho de 2016

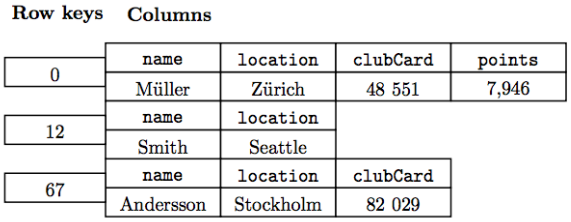

Figura 3: Exemplo de um pequeno banco de dados do cliente. Pode ser visto que as linhas têm chaves únicas que os identifica e que cada linha pode definir seu próprio conjunto de colunas. Note-se que esta é uma imagem simplificada; timestamps são omitidos, e todos os campos são realmente uma sequência de bits. (NÄSHOLM, 2012).

1.3.4 Banco de Daods Orientado a Grafos

A teoria dos grafos serve como suporte para a elaboração dessa base onde, cada vértice representa uma entidade com dados, e as arestas são as relações entre os dados. Pode-se também utilizar o fator da direcionalidade das arestas do grafo como parte do conjunto de dados, no caso do Twitter[1], por exemplo, poderia ser utilizado esse recurso, supondo existirem "João"e "Maria", "João"segue maria no Twitter, ou seja, tem uma aresta na direção de "João"para "Maria"simbolizando essa relação, Caso "Maria"também seguisse "João"haveria outra aresta na direção de "Maria"para "João".

Sua aplicação é muito utilizada em redes sociais e serviços de transito, aproveitando recursos de encontrar caminhos mais curtos entre nós utilizando algoritmos da teoria dos grafos como o de Dijkstra. Além disto é altamente escalável horizontalmente replicando vértices para aumentar a performance e novamente utilizando a teoria dos grafos para particionálo tornando os caminhos das buscas ainda mais curtos. Exemplos de bases: Neo4j[2] e FlockDB[3]

[1]https://twitter.com/, acessado em 30 de Junho de 2016
[2]https://neo4j.com/, acessado em 30 de Junho de 2016
[3]https://blog.twitter.com/2010/introducing-flockdb, acessado em 30 de Junho de 2016

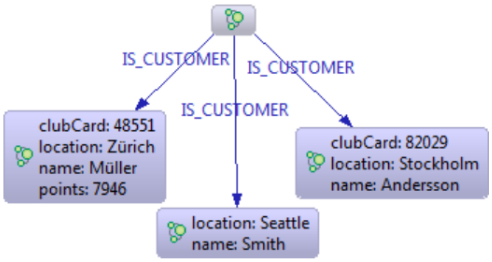

Figura 4: Exemplo de um grafo Neo4j armazenando os dados de 3 clientes. Todos os clientes são alcançáveis a partir do nó vazio. A imagem é um recorte da tela da ferramenta de visualização Neoclipse. (NÄSHOLM, 2012).

Capítulo 2

PostgreSQL e os tipos de dados não convencionais

2.1 PostgreSQL

PostgreSQL(POSTGRESQL, 1996) é um sistema de banco de dados relacional poderoso e open source. Com mais de 15 anos de desenvolvimento tem uma reputação forte sobre sua confiabilidade, integridade dos dados e correção. Ele é executado em todos os principais sistemas operacionais, incluindo Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64) e Windows. É totalmente compatível com ACID, tem suporte completo para chaves estrangeiras, joins, views, triggers e procedures. Inclui a maioria dos tipos de dados SQL:2008: INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, INTERVALO e TIMESTAMP. Também suporta o armazenamento de grandes objetos binários, incluindo imagens, sons e vídeo. Tem interfaces de programação nativas para C / C ++, Java, .Net, Perl, Python, Ruby, Tcl, ODBC, entre outros.

Recentemente também tem adotado algumas funcionalidades NoSQL buscando fornecer mais alternativas reunidas em uma base só, evitando que o serviço tenha que usar uma base SQL para parte de sua funcionalidade e outra base NoSQL para outro grupo de funcionalidades. Dentre elas o suporte orientado a documento(json e xml) e do tipo Chave-Valor(hstore).

Sobre os dados sem schema que utiliza, o Xml é estruturado de maneira hierarquica sendo o padrão W3C standard 1998 o padrão é base para o protocolo de troca de mensagens SOAP, porém seu uso é um pouco complexo e seu desempenho de velocidade é o pior dos dados em armazenamento em documentos postgresql, sem indexação.

Outro tipo é o Hstore, este tipo tem um desempenho bem expressivo pois baseia-se apenas em chave para valores em strings ou outros valores hstore, embora seja eficiente a falta de aninhamento e diferentes tipos dificulta sua utilização para situações mais complexas, não é muito flexível.

Por último temos o tipo JSON e JSONB, que é o tipo Orientado a Documento baseado na notação JavaScript, é utilizado como forma para armazenamento e troca de dados entre várias plataformas web e mobile. Abaixo temos um aprofundamento sobre este tipo por ser base ao trabalho.

2.2 JSON

JSON significa "JavaScript Object Notation"(INTERNATIONAL, 2013), é um formato de dados de fácil leitura e escrita para os humanos e de fácil geração e análise para as máquinas. Um objeto em JSON é um conjunto de pares de chave e valor que podem ser organizados de forma aninhada e(ou) listados em arrays, a chave é uma string e o valor pode ser uma string, um numero, um booleano, um array e até mesmo outro objeto aninhado. O array pode ter valores dos mesmos tipos citados. Pode haver também uma lista de objetos no mesmo nível, esta lista pode obedecer ou não a sua ordem.

2.3 BSON

O termo BSON é uma junção de Binary JSON, ele é a serialização de documentos JSON para binário, foi desenvolvido para melhorar o desempenho na busca das chaves. Além de armazenar os dados em arquivos binários a maior diferença entre ele e o JSON é o uso de tipos de dados com definição de tamanho e tipo pré-determinadas, para armazenar um número inteiro o JSON utiliza uma cadeia de caracteres arbitrária enquanto o BSON utiliza 32 e 64 bits para inteiros e 64 bits para double e floats, tipos booleanos em JSON são representados pela cadeia de caracteres "True"ou "False", em BSON é representado por apenas um bit 0 ou 1. Para valores como cadeias de strings são armazenadas meta-informações como o comprimento dela e, como os outros tipos já tem comprimentos definidos, um parsing na busca por uma chave pode facilmente saltar para os pontos em que interessa no documento.

2.4 Json vs JsonB

Antes do JSONB fazer parte do PosgreSQL Geoffrey Litt desenvolveu em seu projeto final "Improving performance of schemaless document storage in PostgreSQL using BSON"(Aprimorando a performance de armazenamento orientado a documento em PostgrSQL utilizando BSON) (WHITTAKER, 2013) da Yale University(Universidade de Yale), através do fonte do PostgreSQL, bibliotecas de parsing JSON e bibliotecas de parsing BSON do Mongodb, um campo similar chamado BSON, em sua tese pode testar as diferenças de performance destes campos. Posteriormente o PostgreSQL adotou este campo.

Para o primeiro teste utilizou o mesmo documento e variando seu tamanho de 1 à 90 vezes, nele executava uma query simples para encontrar um valor no meio do documento, o teste demonstrou que quanto maior o documento maior era a velozidade do BSON sobre o JSON, com o tamanho máximo o BSON foi 8.28 vezes mais rápido para o JSON.

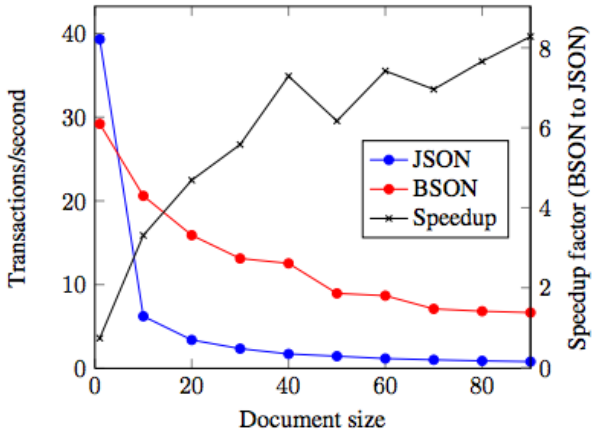

Figura 5: Performance relativa de BSON e JSON para aumento de tamanho do documento sobre número constante de tuplas (WHITTAKER, 2013).

No segundo teste utilizou o mesmo tamanho de documentos mas aumentou o número de tuplas, nele pode constatar que a velocidade de BSON sobre o JSON aumentava com o número de tuplas porém após o número de 100 tuplas essa velocidade começa a convergir para um valor constante de 6 vezes em relação ao JSON.

Outro teste foi realizar queries por uma chave variando seu lugar do começo ao fim do documento e variando o tamanho do documento, a velocidade do bson foi aproximadamente 4 vezes maior em relação ao json para todos os tamanhos de documentos com chaves próximas do começo do documento, para chaves próximas do fim do documento a velocidade do bson foi aproximadamente 6 vezes maior em relação ao json para todos os tamanhos de documentos, isso deve-se à capacidade que o bson tem de atravessar rapi-

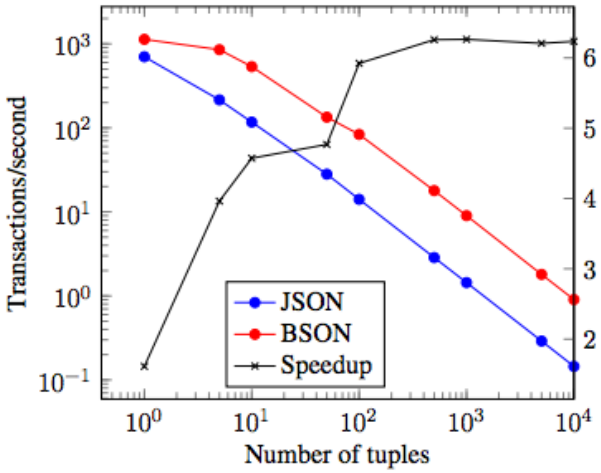

Figura 6: Performance relativa de BSON e JSON para aumento do número de tuplas sobre documento de tamanho estático(WHITTAKER, 2013).

damente as chaves por ter meta-informações sobre seu comprimento enquanto o json tem que analisar caracter a caracter.

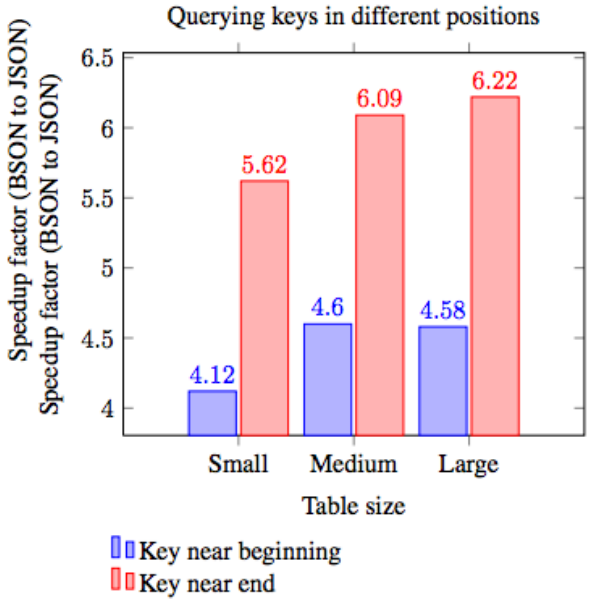

Figura 7: Comparação do fator do aumento de velocidade para chaves perto do começo do documento e perto do fim do documento (WHITTAKER, 2013).

Foi realizado também um teste sobre a composição do documento, ou seja, os tipos de dados dentro do documento, executando queries em 3 documentos, o primeiro com apenas stringsi de 10 caracteres, o segundo apenas com inteiros e o terceiro com apenas booleanos, para strings a valocidade do bson foi 3 vezes superior sobre o json, para inteiros e booleanos a velocidade do bson foi aproximadamente 5,5 vezes superior ao json, o resultado superior de inteiros e booleanos deve-se ao fato do bson utilizar inteiros de 32bits e booleanos de 1

bit, como foi comentado acima, e o json utilizar cadeias de caracteres para representá-los.

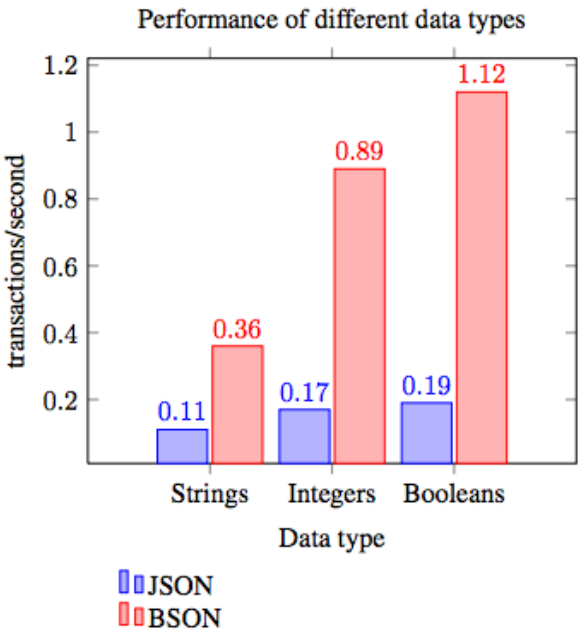

Figura 8: Comparação do fator do aumento de velocidade para documentos contendo exclusivamente um tipo de dado (WHITTAKER, 2013).

Por último foi realizado um teste para verificar o tempo de carregamento dos arquivos em disco, como o bson precisa de um passo a mais no parsing transformando os valores de strings para os tipos de inteiros, booleanos, etc, criando os metadados sobre strings e comprimento das chaves, seu tempo de carregamento é pouco mais demorado em relação ao json, como pode-se verificar na figura 9, entretanto para esse teste foi utilizado o parsing de validação do json e posteriormente foi salvo em disco, caso tivesse sido usado um parsing de validação já em bson essa diferença poderia se igualar ou até mesmo inverter-se.

	1,000 tuples	10,000 tuples	100,000 tuples
JSON	6.100s	62.09s	630000s
BSON	6.139s	62.95s	635600s

Figura 9: Comparação do total de tempo do carregamento entre JSON e BSON para 1.000, 10.000, 100.000 inserções (WHITTAKER, 2013).

2.5 Uso no PostgreSQL

Nesta seção serão abordadas as definições técnicas para utilizar a sintaxe de operadores para executar as tarefas das seções abaixo em campos Json e Jsonb do PostgreSQL, funções que encapsulam funcionalidades não seram abordadas.

2.5.1 Criação de Tabelas

Os tipos Json e Jsonb podem ser criados como um atributo qualquer de qualquer tabela, basta identificar o tipo após o nome do campo:

CREATE TABLE eventos (

nome varchar(200),

attrJson json,

attrBinario jsonb

);

2.5.2 Realizando Inserções

A inserção também ocorre de maneira natural como em outros atributos da tabela, a diferença nesse caso é o conteúdo da coluna:

INSERT INTO eventos values('Show de Sábado', '{"lista de convidados": ["João", "Maria", "Roberto"]}','{"lista de intrusos":[]}');

2.5.3 Execução de Queries

Para execução de queries foram criados operadores adicionais que podem ser utilizados juntos do comando SELECT. Operadores de json e jsonb na tabela

Operador	Operando à Direita	Descrição	Exemplo	Resultado do Exemplo
->	int	Retorna o elemento do array JSON	'["a","b","c"]'::json->2	c
->	texto	Retorna o objeto da chave	'{"a": [1,2,3]}'::json->'a'	[1,2,3]
->>	int	Retorna o elemento do array JSON como texto	'["a","b","c"]'::json->>1	"b"
->>	texto	Retorna o objeto da chave como texto	'{"a": [1,2,3]}'::json->>'a'	"[1,2,3]"
#>	texto[]	Retorna o objeto JSON no caminho	'{"a": {"b":2}}'::json#>'{a,b}'	2
#>>	texto[]	Retorna o objeto JSON no caminho como texto	'{"a": {"b":2}}'::json#>>'{a,b}'	'2'

Tabela 1: Operadores de json e jsonb (POSTGRESQL, 1996)

Capítulo 3

Visualização de Dados

opa (BOSTOCK; OGIEVETSKY; HEER, 2011) Opa (GENGHINI, 2011) Opa (SHIMABUKURO, 2004) Opa

Referências

ARCHIVE, N. *NoSQL Databases*. 2009. Disponível em: ⟨http://nosql-database.org/⟩.

BOSTOCK, M.; OGIEVETSKY, V.; HEER, J. D^3 data-driven documents. *IEEE transactions on visualization and computer graphics*, IEEE, v. 17, n. 12, p. 2301–2309, 2011.

GENGHINI, F. Extensão da plataforma de visualização infovis incorporando recursos de interatividade. Universidade Estadual Paulista (UNESP), 2011.

INTERNATIONAL, E. *Introducting JSON*. 2013. Disponível em: ⟨http://www.json.org/⟩.

NÄSHOLM, P. Extracting data from nosql databases-a step towards interactive visual analysis of nosql data. Chalmers University of Technology, 2012.

POSTGRESQL. *JSON Functions and Operators*. 1996. Disponível em: ⟨https://www.postgresql.org/docs/9.6/static/functions-json.html⟩.

SHIMABUKURO, M. H. *Visualizações temporais em uma plataforma de software extensível e adaptável*. Tese (Doutorado) — Instituto de Ciências Matemáticas e de Computação, 2004.

STRAUCH, C.; SITES, U.-L. S.; KRIHA, W. Nosql databases. *Lecture Notes, Stuttgart Media University*, 2011.

WEBER, S. Nosql databases. *University of Applied Sciences HTW Chur, Switzerland*, 2010.

WHITTAKER, G. L. S. T. J. Improving performance of schemaless document storage in postgresql using bson. 2013.