**UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ**

**MAIRON LUCAS SLUSARZ**

# NEWSQL: CARACTERÍSTICAS E ANÁLISE DA NOVA CATEGORIA DE ARMAZENAMENTO DE DADOS

**PONTA GROSSA**

**2023**

**MAIRON LUCAS SLUSARZ**

# NEWSQL: CARACTERÍSTICAS E ANÁLISE DA NOVA CATEGORIA DE ARMAZENAMENTO DE DADOS

## NewSQL: Characteristics and analisys of the new data storage category

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação da Universidade Tecnológica Federal do Paraná.

Orientador: Tarcizio Alexandre Bini

**PONTA GROSSA**

**2023**

**MAIRON LUCAS SLUSARZ**

# NEWSQL: CARACTERÍSTICAS E ANÁLISE DA NOVA CATEGORIA DE ARMAZENAMENTO DE DADOS

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 07/novembro/2023

---

Tarcizio Alexandre Bini
Doutorado
Universidade Tecnológica Federal do Paraná, Campus Ponta Grossa

---

Richard Duarte Ribeiro
Doutorado
Universidade Tecnológica Federal do Paraná, Campus Ponta Grossa

---

Simone de Almeida
Doutorado
Universidade Tecnológica Federal do Paraná, Campus Ponta Grossa

**PONTA GROSSA**

**2023**

## AGRADECIMENTOS

Suponho não ser capaz de lembrar e expressar gratidão por todos que fizeram parte deste momento da minha vida que também se encerra com este trabalho, que é a vida acadêmica. Porém, espero ser capaz de transparecer a gratidão que sinto neste momento.

Primeiramente, gostaria de agradecer e dedicar este trabalho e meu desempenho acadêmico à meus pais e meu irmão. Agradeço por terem me apoiado de forma incondicional em todos os momentos, desde quando decidi iniciar o curso até os momentos finais. Agradeço pela força e motivação quando pensei em desistir e peço desculpa pela ausência ocasional durante este período.

Também gostaria de agradecer ao meu orientador Prof. Dr. Tarcizio Alexandre Bini, por sua sabedoria e rigidez no processo de pesquisa deste trabalho.

Agradeço também aos meus bons e melhores amigos, que sempre demonstraram apoio nos momentos de dificuldade. E também aos amigos que fiz durante o curso, afinal o caminho teria sido mais complicado e entediante sem o companheirismo e cooperação de vocês.

**RESUMO**

No contexto atual da computação, é crescente o número de aplicações que necessitam de alta vazão de dados. Porém, isto implica geralmente em abdicar de características como a consistência na manipulação de dados. A categoria NewSQL surge como uma alternativa que busca combinar os predicados dos bancos de dados SQL e NoSQL, proporcionando eficiência, escalabilidade e consistência na manipulação dos dados. Porém, esta categoria apresenta carência na definição de suas características, o que dificulta identificar se determinada solução é realmente classificada como NewSQL. Este trabalho se propõe à definir as características essenciais para uma tecnologia fazer parte da categoria NewSQL. Também escolher e avaliar uma solução de armazenamento quanto a estas características. A avaliação se deu pela aplicação de uma ferramenta de *benchmark* em conjunto com códigos escritos em linguagem de programação Python desenvolvidos para observar as características especificadas. As mais evidentes foram o suporte a consultas complexas, onde o SGBD foi capaz de responder a múltiplas junções. Além disso, a solução se destacou por assegurar uma consistência forte, preservando a integridade dos dados mesmo em um cenário de acesso concorrente. Os experimentos evidenciaram que a solução de armazenamento analisada, Google Cloud Spanner, confirma sua classificação como um sistema NewSQL de alta performance e confiabilidade.

Palavras-chave: computação em nuvem; newsql; nosql; relacional.

**ABSTRACT**

In the current context of computing, there is a growing number of applications that require high data throughput. However, this usually implies sacrificing features such as consistency in data handling. The NewSQL category emerges as an alternative that seeks to combine the predicates of SQL and NoSQL databases, providing efficiency, scalability, and consistency in data handling. However, this category lacks a definition of its characteristics, which makes it difficult to identify whether a particular solution is really classified as NewSQL. This work proposes to define the essential characteristics for a technology to be part of the NewSQL category. It also aims to choose and evaluate a storage solution in terms of these characteristics. The evaluation was conducted by applying a benchmark tool in conjunction with Python programming language codes developed to observe the specified characteristics. The most evident were support for complex queries, where the Database Management System was able to respond to multiple joins. In addition, the solution stood out for ensuring strong consistency, preserving data integrity even in a concurrent access scenario. Experiments have shown that the analyzed storage solution, Google Cloud Spanner, confirms its classification as a high-performance and reliable NewSQL system.

Keywords: cloud computing; newsql; nosql; relational.

## LISTA DE FIGURAS

## LISTA DE QUADROS

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| 2PL | *Two phase locking* |
| ACID | Atomicidade, Consistência, Isolamento e Durabilidade |
| AP | *Availability/Partition tolerance* |
| API | *Application Programming Interface* |
| BASE | *Basically Available, Soft State, Eventually Consistent* |
| BASIC | *Basic Availability, Scalability, Instant Consistency* |
| CA | *Consistency/Availability* |
| CAP | *Consistency, Availability, Partition tolerance* |
| CQL | *Cassandra Query Language* |
| CP | *Consistency/Partition tolerance* |
| CPU | *Central Processing Unit* |
| DBaaS | *Database as a Service* |
| DBA | *Database Administrator* |
| GB | Gigabyte |
| JSON | *JavaScript Object Notation* |
| LAN | *Local Area Network* |
| MVCC | *Multi-version concurrency control* |
| NoSQL | *Not Only* SQL |
| OLTP | *Online Transaction Processing* |
| RAM | *Random Access Memory* |
| SaaS | *Software as a Service* |
| SGBD | Sistema Gerenciador de Banco de Dados |
| SGBDD | Sistema Gerenciador de Banco de Dados Distribuído |
| SGBDR | Sistema Gerenciador de Banco de Dados Relacional |
| SQL | *Structured Query Language* |
| TS | *Timestamp* |
| XML | *Extensible Markup Language* |
| WAN | *Wide Area Network* |

**SUMÁRIO**

# 1 INTRODUÇÃO

O armazenamento de dados, embora não seja uma preocupação recente, tem adquirido crescente relevância devido à proliferação da computação em todos os setores da sociedade. Para satisfazer essa necessidade, foram propostos modelos para realizar o armazenamento e gestão de dados ao longo dos anos (BERG; SEYMOUR; GOEL, 2012). Um modelo de armazenamento diz respeito a forma pela qual uma solução representa os dados logicamente, como interpretá-los e tratá-los.

O modelo relacional proposto inicialmente por Codd (1970) ganhou relevância e adeptos desde sua idealização, sendo ainda utilizado com frequência nos dias atuais devido a rigidez na consistência dos dados. Os Sistemas Gerenciadores de Banco de Dados Relacional (SGBDRs) implementam esse modelo e são reconhecidos principalmente por serem dependentes de um esquema, criado durante a modelagem do banco de dados (KHASAWNEH; AL-SAHLEE; SAFIA, 2020). Além disso, possuem uma linguagem de consulta comum chamada SQL (*Structured Query Language*), compatível com todos os SGBRs. Porém, a característica marcante dos SGBDRs é certamente o fato de garantirem as propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade) nas transações submetidas, as quais oferecem confiabilidade nos dados e no processo de gravação destes.

O fato do SGBDR ser estruturado sobre as propriedades ACID limita seu potencial de escalabilidade. Distribuí-lo para operar em vários nós faz com que a efetivação de transações seja um processo lento, afinal todos os nós precisam estar sincronizados para os dados serem consistentes em todas as leituras submetidas. Uma alternativa é escalar o servidor de banco de dados aumentando o seu poder computacional, o que pode elevar razoavelmente o custo do projeto devido ao investimento na aquisição de componentes de *hardware* e *software* (KHASAWNEH; AL-SAHLEE; SAFIA, 2020).

Nos anos 2000, surgiu uma nova classe de sistemas de armazenamento chamada NoSQL (*Not only SQL*), cujo propósito é ser uma solução para aplicações que necessitam de alto *throughput* e que possuem dados que não se encaixam facilmente no modelo relacional (PAVLO; ASLETT, 2016), como aplicações web que são livres de esquema. Essa classe de Sistemas Gerenciadores de Bancos de Dados (SGBDs), por sua vez, não apresenta somente um novo modelo para o armazenamento de dados, mas sim quatro novas abordagens, sendo elas: chave-valor, colunas, grafos e documentos. A maioria dos SGBDs pautados nestes modelos de armazenamento não oferecem as propriedades ACID. Trocam a consistência imediata dos dados (fornecida pelos SGBDRs) pela consistência eventual, na qual nem todos os nós do *cluster* apresentam uma versão atualizada dos dados, sendo sincronizados periodicamente (GROLINGER *et al.*, 2013).

Recentemente, com o avanço de tecnologias de software na área de banco de dados, banda disponível e novos mecanismos de controle de concorrência, surgiu a categoria NewSQL. A mesma agrupa novos SGBDs que aplicam o modelo de armazenamento relacional em uma

arquitetura distribuída, obtendo os benefícios de aumento de disponibilidade de acesso que esta fornece, ao passo que mantém a consistência forte e o uso da linguagem SQL, que são pontos positivos dos SGBDRs. Portanto, o termo NewSQL referencia uma nova categoria e geração de SGBDRs e não uma nova classe em si (MOHMMED; OSMAN, 2017). Essa nova categoria mantém atributos chaves das soluções relacionais e propõe o uso de novas tecnologias, como diferentes mecanismos de controle de concorrência e protocolos de efetivação, uso da memória RAM para armazenamento de toda a base de dados, dentre outras. O resultado são soluções que se desvinculam do padrão que foi adotado nas décadas anteriores, como armazenamento em disco, uso da memória RAM apenas para dados temporários e o emprego de sistemas de *lock* em relações, por exemplo (HARRISON, 2015).

Como apontado por Chereja *et al.* (2021), a categoria NewSQL carece de uma caracterização. Não há um consenso sobre os atributos que uma solução deve atender para ser considerada NewSQL. Também há uma divergência entre o que cada autor considera como sendo características essenciais da categoria NewSQL. Além disso, ferramentas importantes como o DB-Engines não oferecem um filtro para a nova categoria, sendo que os SGBDs identificados como NewSQL nos trabalhos relacionados pesquisados são incluídos no ranking de SGBDRs (DB-ENGINES, 2022).

Este trabalho oferece uma síntese das características essenciais, conforme referenciadas pelos autores da área, para que uma solução seja classificada como NewSQL. Por meio de referencial, serão evidenciadas características obtidas como: suporte a propriedades ACID, suporte à linguagem de consulta SQL, dependência de esquema, *delay* eventual por *lock*, entre outras. Após definidas todas as características fundamentais da categoria NewSQL, será realizada a validação da existência desses atributos em uma solução que supostamente pertence a este grupo. Desta forma, são apresentados critérios de escolha utilizados e como eles levaram à definição do Google Cloud Spanner como SGBD a ser avaliado.

Para avaliação do Google Cloud Spanner frente as características da categoria NewSQL, foi definido um ambiente experimental, composto pela ferramenta de *benchmark* YCSB além de códigos escritos em linguagem Python exclusivamente para a avaliação da solução. Os resultados obtidos demonstraram claramente características NewSQL presentes na solução Google Cloud Spanner como: a presença do modelo relacional, suporte a SQL e suporte a consultas complexas. Os resultados também evidenciam que o Google Cloud Spanner sofre de *delay* por *lock* em situações específicas, como quando o acesso concorrente à tuplas cresce.

## 1.1 Objetivo geral

Agrupar e sintetizar características da categoria NewSQL, analisando-as mediante experimentação em uma solução de armazenamento.

### 1.1.1 Objetivos específicos

- Realizar um estudo para compreender os paradigmas relacional e NoSQL e como eles influenciaram o surgimento do termo NewSQL;
- Definir critérios para pautar a escolha de uma solução NewSQL empregada no ambiente experimental;
- Estabelecer plano de testes para analisar e identificar as características NewSQL presentes no SGBD escolhido;
- Criar o ambiente experimental para atender os requisitos do plano de testes;
- Avaliar os resultados obtidos, constatando como e se as características da categoria NewSQL são respeitadas pela solução de armazenamento avaliada.

## 1.2 Estrutura do trabalho

O presente trabalho está organizado em cinco capítulos. O Capítulo 2 contextualiza e define abordagens para o armazenamento de dados, com ênfase nos modelos relacional, NoSQL e na categoria NewSQL. Neste Capítulo ainda são categorizadas e definidas as características pelas quais as soluções NewSQL são regidas. No Capítulo 3 é apresentada e justificada a escolha da solução NewSQL empregada nos experimentos, assim como as características arquiteturais dela e do *benchmark* aplicado na avaliação do sistema de armazenamento. O Capítulo 4 detalha o ambiente experimental, assim como os resultados obtidos ao analisar a solução NewSQL escolhida frente às características da categoria de armazenamento elencadas. Por fim, o Capítulo 5 discute as conclusões obtidas neste trabalho e aponta possibilidades de trabalhos futuros.

# 2 SOLUÇÕES PARA O ARMAZENAMENTO DE DADOS

A necessidade de armazenar dados não é exclusivamente atual. Desde antes da invenção dos computadores ela existia, e era até então atendida agrupando livros em bibliotecas, por exemplo. Já com o surgimento da computação e dos primeiros códigos de instrução escritos para computadores não foi diferente. Logo no início houve certo grau de manipulação dos dados, por menor que fosse e, em certo momento, esses dados tornaram-se tão densos que foi necessário separá-los do restante do código fonte dos programas (BERG; SEYMOUR; GOEL, 2012). Conforme demanda, surgiram soluções para o armazenamento de dados como o modelo relacional, os modelos da classe NoSQL e a categoria de armazenamento NewSQL, abordados no presente Capítulo.

## 2.1 Solução Relacional

A primeira grande solução para o armazenamento dos dados foi proposta no ano de 1970 por Codd (1970). Neste artigo propôs o Modelo Relacional, o qual armazena os dados utilizando relações. Na base deste modelo estão os conceitos de relações, tuplas e atributos (algumas vezes referenciados como tabelas, linhas e colunas, respectivamente). Segundo a proposta, o usuário seria capaz de recuperar os dados em tuplas obtidas agrupando atributos oriundos de uma relação ou da combinação de relações, as quais são feitas utilizando operadores da álgebra relacional, como junção, união e produto cartesiano, por exemplo. Alguns desses operadores utilizam de atributos chamados de chaves primárias e chaves estrangeiras para construir as tuplas. Codd utilizou de conceitos matemáticos como a teoria dos conjuntos e lógica de predicados para fundamentar sua solução (BERG; SEYMOUR; GOEL, 2012).

Posteriormente, o modelo relacional foi aprimorado por Chen (1976) para abstrair objetos do mundo real em entidades, as quais relacionam-se entre si. Ainda na década de 70, surgiram as primeiras tentativas de transformar as propostas do modelo relacional em uma aplicação, conhecida como SGBDR (BERG; SEYMOUR; GOEL, 2012). Duas propostas acabaram obtendo mais destaque, influenciando em soluções utilizadas nos dias atuais. A IBM desenvolveu um sistema chamado de System R com o objetivo de provar a viabilidade do uso do modelo relacional no mercado. Juntamente com esse sistema, também foi desenvolvida a linguagem de consulta SEQUEL, a qual aproximava a álgebra relacional proposta por Codd (1970) ao idioma inglês, facilitando a curva de aprendizado dos desenvolvedores (CHAMBERLIN, 2012). A outra proposta a ganhar notoriedade surgiu na Universidade de Berkeley, na Califórnia, na ocasião foi desenvolvido um sistema chamado INGRES (o qual posteriormente passou a ser chamado POSTGRES) com um objetivo semelhante ao do System R. Inicialmente utilizava a linguagem QUEL, a qual com o tempo tornou-se obsoleta e, juntamente com a linguagem SEQUEL, contribuíram para a criação da linguagem SQL (BERG; SEYMOUR; GOEL, 2012).

O grande trunfo do modelo relacional é garantir a consistência dos dados a todo momento, pois os SGBDRs que o implementam garantem as propriedades ACID. Mohmmed e Osman (2017) definem este acrônimo da seguinte forma:

- Atomicidade: Garante a completude da transição. Todos os comandos de uma transação devem ser executados sem erros para que esta seja efetivada.
- Consistência: Referencia o estado dos dados antes e depois de uma transação ser executada. Seguindo regras definidas pelo DBA (*Database Administrator*), uma transação deve levar os dados de um estado consistente para outro, sem ir em desacordo com as regras em questão. São exemplos: não permitir um atributo receber um dado de tipo diferente do especificado na criação do esquema e não permitir a exclusão de uma tupla que tem sua chave primária como chave secundária em outras tuplas (conceito de integridade referencial).
- Isolamento: Garante que as transações tenham a "ilusão" de que não há acesso concorrente. Isto é, uma transação não tem acesso a dados que foram alterados porém não efetivados por outra transação.
- Durabilidade: Esta propriedade define que, a partir do momento que uma transação foi dada como concluída, os dados atualizados por ela devem ser persistidos no banco, mesmo em caso de falha.

Essas propriedades permitem a garantia de que os dados obtidos do banco de dados serão consistentes em todas as operações submetidas. Porém, respeitar essas propriedades diminui consideravelmente o potencial de desempenho do SGBDR, visto que propriedades como o Isolamento são alcançadas implementando um mecanismo de bloqueio (*locking*). Isso torna alguns dados indisponíveis para acesso enquanto outra transação o utiliza, reduzindo assim o potencial máximo de operações que podem ser realizadas pelo SGBDR por unidade de tempo (*throughput*) (STONEBRAKER, 2010). Dada essa limitação, aliada ao surgimento de aplicações que submetem muitas requisições em um curto espaço de tempo, surgiu espaço para a pesquisa e criação de novos modelos de armazenamento, discutidos na seção 2.2.

## 2.2 Solução NoSQL

Por volta dos anos 2000, popularizou-se no mundo inteiro o uso da internet, a qual está cada vez mais presente no dia a dia das pessoas. Com essa revolução, também nasceu uma nova demanda quanto ao armazenamento de dados. Até então os aplicativos consumiam dados somente localmente e, mesmo em casos onde havia mais de um aplicativo utilizando o banco ao mesmo tempo, a perda de desempenho não era um problema grave. Porém, com o uso da internet, aplicativos começaram a requisitar dados remotamente ao SGBDR (BERG; SEYMOUR; GOEL, 2012). Como o número de usuários submetendo transações ao mesmo tempo cresceu, os SGBDRs começaram a demonstrar dificuldade em atender todas as requisições em tempo hábil, pois isso requer uma arquitetura capaz de dar vazão a um número crescente de

requisições paralelas. Isto exigiu em muitos casos a migração para uma arquitetura distribuída, a qual não é facilmente alcançada no modelo relacional por conta da necessidade de consistência dos dados (HARRISON, 2015). Visando suprir essa demanda de vazão de dados, surgiu uma nova classe de SGBDs nomeada NoSQL (Not Only SQL), a qual define uma nova gama de modelos para armazenamento de dados flexíveis a receber diferentes tipos de dados, não sendo dependentes de esquema.

Schreiner *et al.* (2019) pontuam que SGBDs NoSQL obtem êxito quando utilizados pela nova geração de aplicações, como sistemas de finanças e jogos online por exemplo. Pois esses caracterizam-se por ter um grande número de usuários, gerando um elevado tráfego de dados e múltiplas transações OLTP (*Online Transaction Processing*) com um tempo de vida curto e que não realizam o processamento e análise de grandes massas de dados.

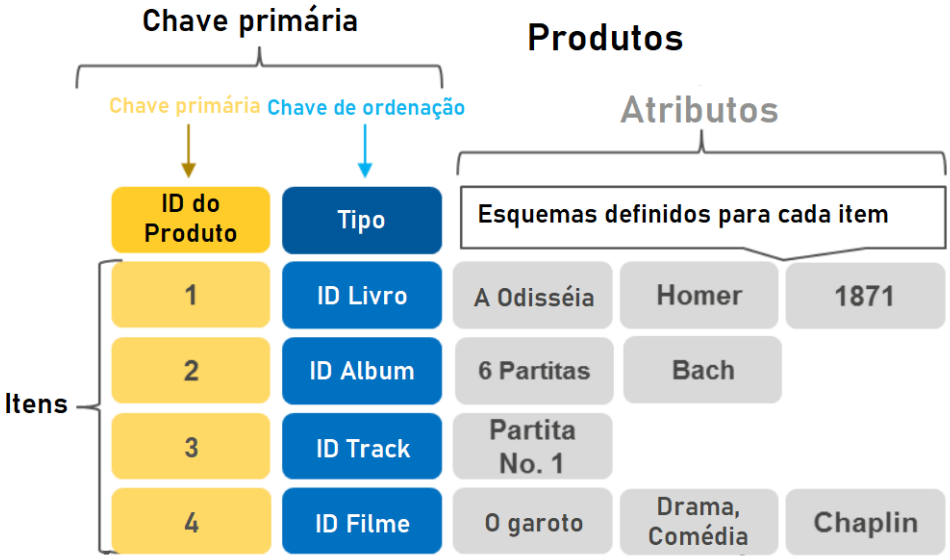
Grolinger *et al.* (2013) afirmam que o acrônimo NoSQL sugere que o uso da Linguagem de Consulta Estruturada (SQL) não é o objetivo crucial da classe, porém algumas soluções também fazem uso de uma linguagem para manipulação e recuperação dos dados. Um exemplo é o SGBD Cassandra, que possui uma linguagem própria chamada CQL (*Cassandra Query Language*), a qual apesar de apresentar semelhanças de sintaxe com o SQL, possui particularidades para elevar o potencial de desempenho e escalabilidade da solução (CHEBOTKO; KASHLEV; LU, 2015). Além disso, a maior parte das aplicações NoSQL não garantem as propriedades ACID. Geralmente são pautadas sobre o acrônimo BASE, o qual é definido por Chaudhry e Yousaf (2020) como:

- *Basically Available*: Garante que os dados sempre estarão disponíveis e toda requisição terá uma resposta por parte do SGBD.
- *Soft State*: Essa propriedade relaciona-se diretamente com a consistência eventual também definida no acrônimo. *Soft State* significa que o estado dos dados armazenados no SGBD pode mudar sem que aconteça uma operação de escrita. Sendo que essa atualização do estado pode acontecer porque eles expiraram ou porque o SGBD iniciou um procedimento de sincronização dos nós do *cluster*, atualizando e mudando o estado dos dados.
- *Eventually Consistent*: Significa que o estado de consistência será alcançado em algum momento, quando todas as réplicas do *cluster* serão sincronizadas. Esse processo ocorre de tempos em tempos, por isso nem todos os nós apresentam dados consistentes a todo momento.

Portanto, utilizando as propriedades BASE, a consistência imediata dos dados (característica relacional) é substituída pela consistência eventual. Desse modo, o banco está sempre disponível para acesso, visando assim aumentar o seu *throughput*. Além disso, o banco pode ser escalonado pela adição de novas máquinas ao *cluster* do SGBD de forma mais fácil, pois não precisa manter os dados consistentes todo o tempo em todos os nós (GROLINGER *et al.*, 2013). Quanto ao modelo de armazenamento dos dados, de acordo com Sadalage e Fowler (2013), a classe NoSQL pode ser classificada em quatro modelos, sendo eles:

- Chave-valor: nesse modelo um valor ou um conjunto de valores são associados a uma chave única e atômica, a qual é utilizada para recuperar os mesmos. O valor referenciado a uma chave pode ser um dado ou uma chave que referencia outros valores. Segundo Grolinger *et al.* (2013), como o campo de valor suporta vários formatos de dados, o modelo chave-valor é considerado livre de esquema. Além disso, esse modelo é notavelmente eficiente quando aplicado no armazenamento de dados distribuídos, por ser necessário apenas uma chave para recuperar o valor e não necessitar de agrupamento de dados. Porém não é adequado para casos nos quais os dados mantém relacionamentos entre si ou possuem estrutura complexa (uma aplicação orientada a objetos, por exemplo). Os pares de chave-valor são distribuídos no *cluster* utilizando uma tabela *hash* (RASHEED; QUTQUT; ALMASALHA, 2019). Um exemplo de como os dados são armazenados no SGBD DynamoDB (que utiliza o modelo chave-valor) pode ser observado na Figura 1.

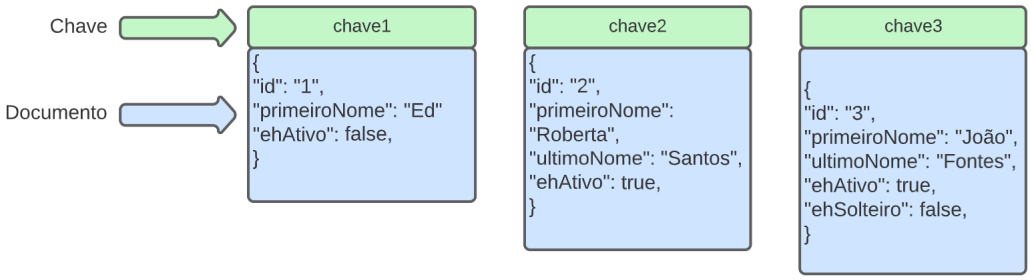
**Figura 1 – Armazenamento chave-valor do SGBD DynamoDB**




**Fonte: Adaptado de DynamoDB (2022)**

- Documentos: modelo semelhante ao esquema chave-valor, utiliza chaves únicas para referenciar os documentos. Estes, por sua vez, geralmente são representados usando JSON (*JavaScript Object Notation*) ou algum formato semelhante, como XML (*Extensible Markup Language*). O uso de documentos permite criar estruturas de dados relativamente complexas, como aninhamento de objetos por exemplo, ao mesmo tempo em que não necessita de esquema (GROLINGER *et al.*, 2013). A Figura 2 exemplifica a forma que o modelo de documentos armazena dados.
- Família de colunas: no modelo relacional, os dados são gravados em linhas, portanto é necessário o uso de operadores para produzir uma tupla. O modelo família de colunas, por sua vez, armazena os dados em colunas, buscando alocar todos os dados da

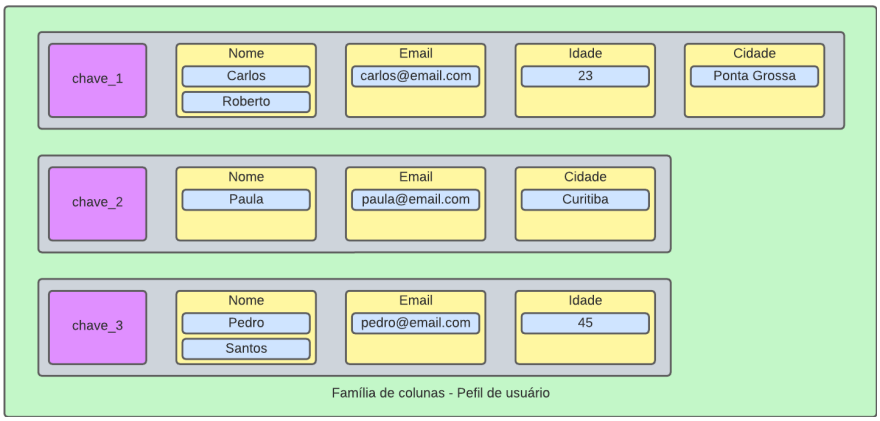
**Figura 2 – Armazenamento por documentos**




**Fonte: Autoria própria (2023)**

tupla do modelo relacional em uma mesma coluna. Essa organização permite que as operações de agregação, por exemplo, sejam otimizadas, porque os dados estarão localizados em um mesmo bloco do disco (HARRISON, 2015). A Figura 3 ilustra como o modelo orientado a colunas armazena os dados, exemplificando uma situação de perfil de usuário.

**Figura 3 – Armazenamento por família de colunas**

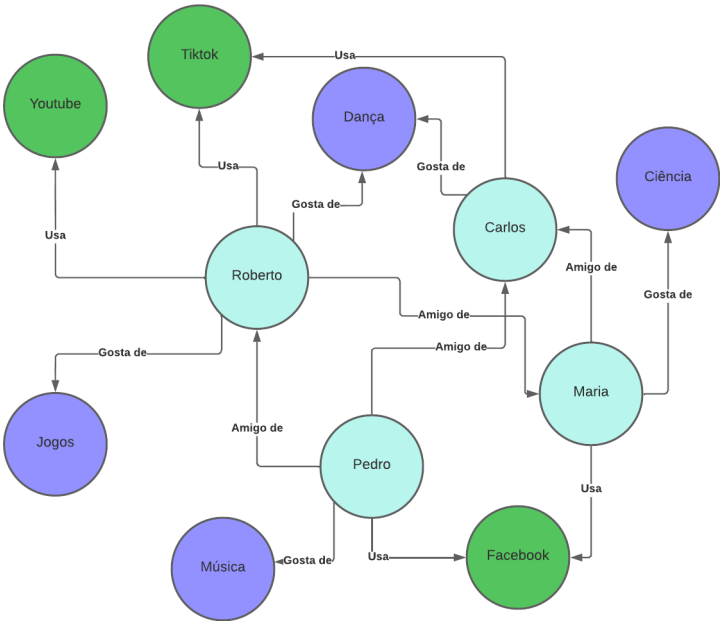



**Fonte: Autoria própria (2023)**

- Grafos: diferente dos demais modelos NoSQL apresentados, o armazenamento que utiliza-se de grafos é fundamentado em fortes princípios matemáticos. Neste modelo, os vértices são tratados como nós e as arestas como relacionamentos, tornando possível criar um esquema de relacionamento entre os dados armazenados nesses nós. Além disso, pode utilizar-se de algoritmos já consolidados de busca, remoção e adição de vértices em grafos para se obter ganho de desempenho (HARRISON, 2015). A Figura 4 ilustra um grafo com informações sobre preferências de usuários.

Enquanto SGBDs Relacionais não possuem potencial de desempenho para atender aplicações do tipo OLTP sem gerar lentidão, SGBDs NoSQL distribuídos que priorizam o *throughput* não atendem os requisitos de aplicações que necessitam de dados consistentes a todo o momento. Quando uma aplicação precisa tanto de alto *throughput* quanto de dados consistentes, muitas vezes os mesmos precisam ser distribuídos em diferentes SGBDs. De acordo com cada modelo de armazenamento empregado, os SGBDs são capazes de fornecer características

**Figura 4 – Armazenamento utilizando grafos**




**Fonte: Autoria própria (2023)**

específicas de desempenho ou consistência. Essa técnica recebe o nome de persistência poliglota, a qual permite associar múltiplos modelos de armazenamento de dados em uma mesma aplicação (ZDEPSKI, 2019). Porém, as diferenças que existem entre os modelos de armazenamento tornam complexa a integração entre os dados, que muitas vezes precisa ser realizada pela própria aplicação. Por isso, surge espaço para o desenvolvimento de tecnologias capazes de integrar desempenho e consistência, abordadas na seção 2.3.

## 2.3 Solução NewSQL

Buscando agrupar SGBDs que utilizam novas tecnologias de software, principalmente quanto a mecanismos de controle de concorrência, distribuição de dados em um servidor e protocolos de efetivação de transações usando rede, Aslett (2011a) cunhou uma nova categoria de SGBDs chamada NewSQL. O objetivo desta categoria é garantir os mesmos benefícios de ambas as classes abordadas: representar os dados utilizando o modelo de armazenamento relacional, fornecendo a consistência imediata dos SGBDRs e a possibilidade de suportar instruções SQL. Ao mesmo tempo que possui uma arquitetura distribuída com alto *throughput*, de forma semelhante aos SGBDs NoSQL (GROLINGER *et al.*, 2013).

A definição do que é necessário para ser membro da categoria NewSQL causa controvérsia entre os autores que abordam o tema, sendo que, pouco depois de publicar o artigo no qual pela primeira vez apresentou a categoria NewSQL, Aslett (2011b) publicou um novo artigo discutindo o que de fato pode ser o NewSQL. Nesse material, ele afirma que NewSQL nada mais é que “[...] nossa abreviação para os vários novos fornecedores de bancos de dados escaláveis e de alto desempenho.” (ASLETT, 2011b). O autor alerta sobre a interpretação do

acrônimo: Assim como no caso do NoSQL que não necessariamente se desassocia do SQL, o “novo” do NewSQL diz respeito a novos SGBDRs e não a um novo SQL. Em Aslett (2021), o primeiro autor a falar sobre NewSQL voltou a afirmar que “[...] definir uma nova categoria de mercado não era nossa intenção. [...]” e que na realidade estavam apenas usando esse termo para referenciar um grupo de novos SGBDs. Grupo o qual “[...] vários produtos inclusos nele são vagamente afiliados e o que tem em comum é o fato de combinarem os benefícios do modelo relacional e das arquiteturas distribuídas.” (ASLETT, 2021). Portanto, é necessário definir como os produtos são afiliados e agrupados a essa categoria, sendo que isso é feito através da definição das características comuns a esses produtos.

### 2.3.1 Características

É difícil explicitar quais as características que definem um SGBD como integrante da categoria NewSQL. Por exemplo, um SGBDR é facilmente identificado por ter suporte a linguagem SQL, ser dependente de esquema e principalmente pelo uso do modelo relacional como formato de armazenamento dos dados. Os SGBDs NoSQL por sua vez, são categorizados pelo modelo de armazenamento utilizado: colunas, grafos, chave-valor e documentos. Já a categoria NewSQL, como destacam Pavlo e Aslett (2016, p. 53): "não são um desvio radical das arquiteturas existentes, mas sim representam um novo capítulo no desenvolvimento contínuo de tecnologias de bancos de dados."

A categoria NewSQL, portanto, agrupa características e tecnologias já conhecidas dos SGBDRs, como o modelo relacional de armazenamento e o suporte a manipulação dos dados utilizando comandos SQL. Porém a forma como os SGBDs NewSQL tratam a consistência, o acesso aos dados e a recuperação de falhas são aspectos variáveis de implementação para implementação (KNOB *et al.*, 2019). Como destacado por Mohmmed e Osman (2017), NewSQL não representa uma nova classe e um novo modelo de armazenamento de dados e sim representa uma nova categoria da classe Relacional, englobando os SGBDRs que operam sobre uma arquitetura distribuída. Por isso, é necessário definir os critérios a serem atendidos para ser considerado um SGBD da categoria NewSQL.

Segundo Pavlo e Aslett (2016), um SGBD NewSQL precisa atender a alguns requisitos essenciais: (1) possuir um sistema de controle de concorrência que permita que mais de um usuário execute leitura em uma relação ao mesmo tempo e essa leitura seja de um dado consistente. O controle de concorrência é, na maioria das aplicações, realizado por um sistema MVCC (*Multi-version concurrency control*) que armazena mais de uma versão de uma mesma tupla relacionando-a ao exato momento em que ela sofreu alteração. (2) Ser capaz de suportar a utilização e criação de índices secundários nas relações, visando recuperar dados com mais velocidade e interagindo com porções menores de dados. (3) Possuir uma arquitetura *shared nothing*, na qual cada nó do *cluster* é capaz de executar uma operação sem requisitar recursos de um nó mestre ou outro nó do *cluster*. Ou seja, cada nó tem para si o próprio CPU, memó-

ria RAM e dispositivos de disco (HARRISON, 2015). Em caso de transações que precisem de dados presentes em nós diferentes do *cluster*, essa transação é dividida em partes que são executadas por cada nó separadamente. Outra exigência citada por Pavlo e Aslett (2016) é que um SGBD NewSQL precisa suportar a execução de comandos SQL.

Para Kumar e Charu (2014), a categoria NewSQL engloba vários SGBDs desenvolvidos para trazer os benefícios de um SGBDR a uma arquitetura distribuída. Sendo assim, projetados para suportar a linguagem SQL enquanto mantém escalabilidade e performance. O autor também cita as principais características de um sistema NewSQL:

1- Oferecer interação utilizando comandos SQL;

2- Suportar as propriedades ACID para transações;

3- Utilizar um sistema de *non-locking* para controle de concorrência;

4- Oferecer performance por nó maior do que soluções relacionais;

5- Suportar uma arquitetura escalonável, paralelizada, *shared nothing*. Capaz de executar em vários nós sem sofrer gargalo;

Ismail *et al.* (2021) afirmam que o termo NewSQL é usado para distinguir um grupo de SGBDs capazes de ter a tolerância a falhas e a escalabilidade dos NoSQL enquanto mantém o modelo relacional e a consistência forte dos SGBDRs. Caracterizam os SGBDs NewSQL como ideais para aplicativos que: "(1) incluem transações de tempo de execução curto (por exemplo, *INSERT, UPDATE, DELETE*), (2) acessam uma pequena quantidade de dados de todo o banco de dados usando pesquisa em índices."(ISMAIL *et al.*, 2021).

Ismail *et al.* (2021), baseados em outros trabalhos, classificam os SGBDs NewSQL em quatro categorias:

- Novas arquiteturas: SGBDs desenvolvidos desde o princípio para atender uma performance escalável. Projetados para ter uma arquitetura completamente distribuída, capaz de operar em vários nós. Tendo ainda, nativamente, mecanismos de tolerância a falhas, replicação e controle de concorrência *lock-free*, permitindo a execução de transações paralelas em múltiplos nós do *cluster*. Além disso, os nós operam em *shared-nothing*.
- Novos mecanismos de armazenamento MySQL: Essa categoria agrupa SGBDs que implementam mudanças no MySQL para atender a aplicações OLTP. Essas novas versões do MySQL utilizam mecanismos de distribuição dos dados e de consultas para melhorar o desempenho.
- *Middleware* de *cluster/sharding* transparente: Um SGBD dessa categoria dá ao usuário e a aplicação a ilusão de que opera em uma unidade centralizada. Porém, como em outros SGBDs NewSQL, na realidade funciona em um sistema distribuído. Os SGBDs dessa categoria utilizam a abordagem relacional já conhecida enquanto dividem o banco de dados em porções menores. A principal vantagem é a possibilidade de reutilizar os códigos do lado da aplicação que eram empregados na comunicação com o SGBDR sem necessidade de alterações profundas.

- *Cloud Database as a Service*: Como o nome sugere, essa categoria utiliza-se dos conceitos de computação em nuvem para construir uma solução NewSQL. Em geral, não há grandes diferenças de arquitetura e funcionamento entre os SGBDs que operam em servidores dedicados e os que operam em nuvem.

Por fim, a classificação de Ismail *et al.* (2021) difere da de Pavlo e Aslett (2016) por exemplo, que não considera implementações que apresentam melhorias no SGBD MySQL para ser utilizado em aplicações OLTP como pertencentes à categoria NewSQL. Os demais trabalhos analisados (CHAUDHRY; YOUSAF, 2020; CHEREJA *et al.*, 2021; GROLINGER *et al.*, 2013; HARRISON, 2015; KNOB *et al.*, 2019; KUMAR; CHARU, 2014; SCHREINER *et al.*, 2019) não aprofundam a questão da divisão dos SGBDs NewSQL em categorias, porém na maioria dos casos são utilizadas as mesmas categorias definidas por Pavlo e Aslett (2016) (novas arquiteturas, *middleware* transparente, *database as a service*). Como Pavlo e Aslett (2016) e Ismail *et al.* (2021) definem categorias para os SGBDs NewSQL de acordo com aspectos técnicos e diferenças de arquitetura (principalmente quando definem a categoria "novas arquiteturas"), compreender as particularidades com relação ao comportamento em transações, ao controle de concorrência, assim como a forma pela qual manipulam e armazenam os dados é um procedimento importante para entender o funcionamento das soluções NewSQL.

Com relação a forma como os dados são tratados, Schreiner *et al.* (2019) afirmam que, apesar de possuírem algumas características em comum, como o controle de concorrência *lock-free* e uma arquitetura *shared-nothing*, cada solução NewSQL possui uma estratégia particular para manipulação dos dados. Pontuam também que "SGBDs NewSQL geralmente são bancos de dados *in-memory*", o que significa que buscam utilizar a memória RAM como armazenamento principal, evitando o custo associado a lentidão apresentada pelos discos rígidos. Os SGBDs que implementam tal abordagem utilizam-se de algoritmos para identificar tuplas pouco acessadas e transferi-las para um armazenamento secundário, dessa forma suportam uma quantidade de dados maior do que o tamanho da memória RAM, além do fato de pulverizarem os dados em vários nós conectados a um sistema distribuído.

Pavlo e Aslett (2016) destacam que a ideia de distribuir os dados de um SGBD em partições não é exatamente nova e que desde a década de 80 tentativas de fazê-lo aparecem no meio acadêmico. Porém, esses projetos acabaram não obtendo sucesso principalmente por conta da limitação tecnológica da época, assim como o elevado custo dos componentes de computador disponíveis. Porém, nos dias atuais, os dois pontos não são mais um empecilho, além do fato de que agora existe demanda para esse tipo de solução, por conta do alto tráfego de dados gerado pelas aplicações web e mobile.

Quanto a como os dados são pulverizados através do *cluster*, tanto Schreiner *et al.* (2019) quanto Pavlo e Aslett (2016) destacam que as relações são divididas horizontalmente em fragmentos, chamados também de partições. O SGBD atribui as tuplas para cada fragmento do *cluster* aplicando uma função baseada nos valores de uma ou mais colunas em específico. Esses valores são chamados de atributos de particionamento (PAVLO; ASLETT, 2016). Através

disso, cada tupla é atribuída a um fragmento de acordo com um particionamento de hash ou intervalo aplicado aos atributos. Cada nó armazena fragmentos co-relacionados, com o objetivo de que uma transação possa ser satisfeita unicamente por uma máquina local, sem necessidade de comunicar-se com outros nós do *cluster* (SCHREINER *et al.*, 2019). Quando uma transação precisa acessar dados de outros nós, as soluções NewSQL aplicam protocolos para garantir que uma transação só seja concluída se obter sucesso em todos os nós envolvidos, caso contrário ela é revertida em todos os fragmentos.

Mesmo que uma transação qualquer seja completada utilizando apenas um nó do *cluster*, ainda assim é necessário considerar a comunicação entre nós, pois, assim como nos SGBDs NoSQL, a maioria dos SGBDs NewSQL aplicam replicação de dados para aumentar a disponibilidade do banco. Por isso, é necessário tratar tanto da replicação em si como com a atualização constante dessas réplicas, pois os dados precisam estar consistentes. Pavlo e Aslett (2016) destacam que os SGBDs NewSQL optam pela chamada consistência forte, a qual implica que todas as réplicas devem ter conhecimento do sucesso de uma transação que realiza escrita. A vantagem no uso da consistência forte é que transações subsequentes sempre terão acesso ao dado mais recente, independente do nó em que realizarem a leitura. A desvantagem, porém, está no fato de que é necessário o uso de um protocolo de comunicação constante entre os nós. A necessidade do uso de tal protocolo pode levar a lentidão, devido a falha em uma das réplicas ou até mesmo a falha de conexão entre os nós envolvidos. Posteriormente, serão abordados exemplos de tais protocolos na subseção 2.3.2 e na subseção 2.3.3.

Em relação a como os dados são replicados, Pavlo e Aslett (2016) mostra que existem duas formas. A primeira, chamada ativo-ativo, consiste em todas as réplicas executarem a requisição ao mesmo tempo, de forma paralela. O problema desse método é que o *delay* de chegada de uma transação às diversas réplicas pode resultar em dados diferentes entre os nós, pois transações submetidas em tempo semelhante ao SGBD podem ser executadas em ordem diferente nos nós. O segundo método, chamado ativo-passivo, consiste em um nó executar a transação e transmitir o resultado para as réplicas, evitando assim a inconsistência, ao custo que pode gerar lentidão e um alto tráfego de dados entre os nós. A maioria dos SGBDs NewSQL implementam o método ativo-passivo, pois prezam pela consistência dos dados (PAVLO; ASLETT, 2016). Geralmente as réplicas ficam em um mesmo local, utilizando comunicação em LAN (*Local area network*), a qual apresenta menos problemas de lentidão que uma conexão WAN (Wide area network), na qual os nós podem estar isolados por grandes distâncias geográficas (SCHREINER *et al.*, 2019). Os SGBDs NewSQL podem ser configurados para realizar replicação através da rede WAN, que é um aspecto que os diferencia se comparado aos SGBDs relacionais e algumas soluções NoSQL. Porém, como destacado por Pavlo e Aslett (2016), essa escolha inevitavelmente causa lentidão na execução das operações no SGBD NewSQL, pelo proporcional aumento do tempo de comunicação associado ao aumento da distância entre os nós, além do fato da conexão WAN ser mais suscetível a rompimentos na comunicação.

Quanto ao modelo de armazenamento utilizado pelas soluções NewSQL, Chereja *et al.* (2021) pontuam que todas elas são construídas para suportar o modelo relacional, porém algumas são bancos de dados multi-modelos. Esse termo pode ser interpretado de duas maneiras. Na primeira forma, o SGBD é capaz de suportar outros tipos de dados nativamente, como o SGBD SAP-HANA (2022) que é capaz de possuir simultaneamente instâncias de um banco de dados relacional e um orientado a grafos, permitindo a troca de dados entre os dois modelos.

A outra forma de operação que caracteriza um SGBD como multi-modelo acontece quando o SGBD NewSQL trata os dados como relacionais, criando-os através de comandos SQL, mas os armazena utilizando outro modelo, como o chave-valor. Um exemplo desse caso é detalhado por Lifhjelm (2021), o qual aborda o funcionamento do SGBD CockroachDB (2022). Neste SGBD, cada nó possui várias camadas, incluindo a camada SQL e a camada transacional chave-valor. A primeira camada realiza a comunicação com a aplicação, recebendo os comandos SQL e os convertendo para uma sequência de comandos de leitura e escrita equivalentes para serem submetidos ao modelo chave-valor. Esses comandos são então enviados para a camada transacional, a qual é responsável por organizá-los de modo a garantir a consistência, atomicidade e isolamento dos dados. Por fim, essas operações são repassadas para a camada que efetivamente armazena os dados, no formato chave-valor. Ela então executa os respectivos comandos e devolve uma resposta, a qual é convertida para uma saída compatível com o modelo relacional novamente na camada SQL. Dessa forma, mesmo utilizando modelos auxiliares, o objetivo dos SGBDs NewSQL é trabalhar primariamente com a linguagem SQL.

Sobre a aplicação da linguagem SQL, Chereja *et al.* (2021) afirmam que nenhuma das soluções NewSQL tem uma visão sistemática sobre ela. O grau de suporte é variável de implementação para implementação, pontuando que, mesmo com um suporte reduzido, a capacidade de utilizar SQL é uma vantagem para essa categoria pela facilidade de adaptação do usuário, que geralmente já está familiarizado com a linguagem de consulta. Chereja *et al.* (2021) destaca ainda que nem todos os SGBDs NewSQL aplicam o SQL por completo, como é o caso do SGBD VoltDB, o qual não suporta a criação de chaves estrangeiras. Essas particularidades quanto a aplicação da linguagem SQL tem consequências na forma como as aplicações NewSQL tratam às transações que os alteram.

Relacionado ao comportamento em transações, Chaudhry e Yousaf (2020) definem os SGBDs NewSQL como sendo BASIC (*Basic Availability, Scalability, Instant Consistency*). *Basic availability* indica que o sistema sempre irá responder a consultas de leitura/escrita. *Scalability* garante que novos nós podem ser adicionados à medida que a carga de trabalho aumenta. Por fim *Instant Consistency* garante que “se transações de escrita e leitura são realizadas consecutivamente, então o resultado retornado pela operação de leitura deve ser o mesmo que o escrito pela operação de escrita” (CHAUDHRY; YOUSAF, 2020). Essa classificação complementa a característica dos SGBDs garantirem as propriedades ACID, adicionando elementos da arquitetura distribuída, como disponibilidade e escalabilidade.

Quanto às propriedades ACID nas transações, Chereja *et al.* (2021) complementam que não há uma regra geral usada pelos SGBDs NewSQL para garanti-las, porém, Sivakumaran e Ali (2017) tenta fornecer uma generalização:

> Atomicidade é garantida em todas as situações, incluindo falhas de energia, erros e travamentos. A propriedade de consistência garante que qualquer transação mudará o SGBD de um estado para outro. A propriedade de isolamento é para garantir que as transações possam ser executadas concorrentemente. O isolamento é obtido através do controle de concorrência. Quando uma transação é validada, a durabilidade é mantida.(SIVAKUMARAN; ALI, 2017, p. 2)

Depois de analisar os aspectos que compõem a arquitetura das soluções relacionais, NoSQL e NewSQL, conclui-se que definir quais são as características fundamentais das soluções NewSQL é uma tarefa complexa. Isso ocorre pois cada aspecto dessa arquitetura pode ser implementada por diferentes algoritmos. Porém, para facilitar a organização e a compreensão deste trabalho, o Quadro 1, pautado na revisão bibliográfica realizada, faz um levantamento do que é consenso entre autores, no que diz respeito a características fundamentais dos três formatos de armazenamento elencados.

**Quadro 1 – Comparativo de características das soluções de armazenamento**

| **Características** | **Relacionais** | **NoSQL** | **NewSQL** |
|---|---|---|---|
| Propriedades ACID | Fornece | Não fornece | Fornece |
| Modelo de armazenamento | Relacional | Colunas, chave-valor, grafos, documentos | Relacional, multi-modelos |
| Processamento OLTP | Ineficiente | Eficiente | Eficiente |
| Suporte a SQL | Sim | Não | Sim (Variável) |
| Escalabilidade horizontal | Não | Sim | Sim |
| Naturalmente distribuído | Não | Sim | Sim |
| Suporte a consistência forte | Sim | Não | Sim |
| Depende de esquema | Sim | Não | Sim |
| Suporte a alto fluxo de dados | Baixo | Bom | Bom |
| *Delay* por *lock* | Sim | Não aplica | Eventualmente |
| Suporte a consultas complexas (junções, por exemplo) | Sim | Não | Sim |

**Fonte: Autoria própria (2023)**

Observando o Quadro 1, é possível constatar características de destaque da categoria NewSQL, como a capacidade de fornecer consistência forte ao mesmo tempo que é horizontalmente escalável. A capacidade de fornecer essas características inicialmente conflitantes (no modelo relacional), deve-se principalmente a aplicação de protocolos de efetivação e de controle de concorrência. Portanto é importante entender o que são, como funcionam e como esses protocolos são capazes de unir as duas características em uma solução. Como base arquitetural das soluções NewSQL, a literatura (CHAUDHRY; YOUSAF, 2020; ISMAIL *et al.*, 2021;

PAVLO; ASLETT, 2016; SCHREINER *et al.*, 2019) destaca dois algoritmos para implementação de protocolos, sendo o *Multi-version concurrency control* (MVCC) e o *two phase locking* (2PL), abordados respectivamente na subseção 2.3.2 e na subseção 2.3.3.

### 2.3.2 Multi-version concurrency control

A forma mais simples de garantir a consistência dos dados em um SGBD é através do bloqueio de tuplas (algoritmo de *locking*). Porém bloquear uma tupla faz com que outras transações não sejam capazes de realizar operações de leitura ou escrita sobre a mesma, reduzindo assim o *throughput* do sistema ao mesmo tempo que mantém consistência (HARRISON, 2015). Com o crescimento do número de aplicações web, cresceu também o uso de SGBDs distribuídos. Em alguns casos, essas aplicações possuem funcionalidades que necessitam de dados consistentes, principalmente quando manipulam dados sensíveis, como a quantidade de produtos disponíveis ou dados que envolvem transações financeiras por exemplo.

Com relação aos tipos de transações que são frequentemente submetidas ao SGBD, Wei *et al.* (2021) ao analisar o resultado do *benchmark* TPC-E (TPC, 2015) operando sobre um modelo de bolsa de valores destaca que: "79% das transações são do tipo somente leitura em tempo de execução". Esse fato traz um problema caso o SGBD utilize o algoritmo de *locking*, pois transações que realizam somente operações de leitura reduzem a concorrência do banco de dados. Isso ocorre porque essas consultas são realizadas geralmente através da junção de relações, que é uma operação com potencial para gerar lentidão. Afinal, essa operação pode ser dividida em várias sub-consultas e essas sub-consultas podem demorar para concluir por esperarem um dado ser liberado por outra transação que está sendo executada concorrentemente. Como todas as sub-consultas precisam ser executadas por completo para que a consulta inicial seja respondida, há um potencial *delay*. Uma vez que esse tipo de operação não altera o dado presente no banco, várias transações do tipo somente leitura poderiam ser realizadas de forma concorrente sem comprometer a consistência dos dados.

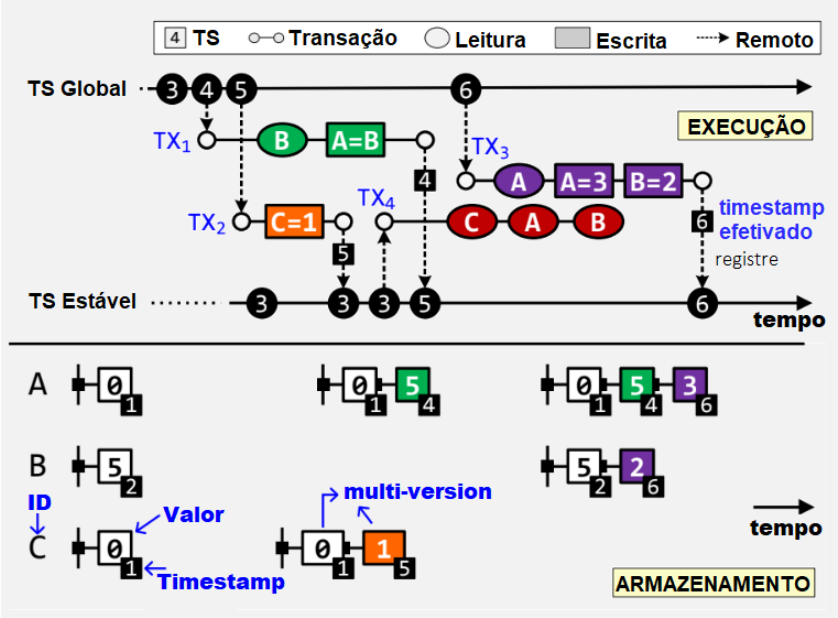
Para contornar a limitação da vazão de dados nas consultas, foi proposto o protocolo MVCC. Esse protocolo consiste em um algoritmo que não sobrescreve uma tupla quando ela é atualizada por uma transação, mas sim armazena a versão atual da tupla e cria uma nova versão dela com o dado atualizado. Todas as tuplas recebem um campo adicional do tipo *timestamp* referente ao momento em que aquela versão foi efetivada. Dessa forma, o SGBD mantém várias versões de uma mesma tupla, permitindo assim que uma transação de somente leitura possa ser completada mesmo que a tupla sofra escrita por outra transação (WEI *et al.*, 2021). Além disso, o fato de armazenar várias versões permite que operações consigam acessar tuplas antigas, facilitando também o processo de recuperação de falhas (WU *et al.*, 2017).

Segundo Wei *et al.* (2021), a versão mais simples do protocolo MVCC suporta operar em um *cluster* fazendo uso de um "sequenciador centralizado" (por vezes identificado como oráculo) o qual é responsável por fornecer o *timestamp* para as transações que realizam escrita,

de modo que essas possam ser realizadas de forma coordenada. Wei *et al.* (2021) descrevem ainda o funcionamento do MVCC em conjunto com o protocolo *two phase locking* (o qual será abordado na subseção 2.3.3). O oráculo possui dois *timestamps* diferentes: um é usado por transações de leitura-escrita (Global) e outro é usado por transações de somente leitura (Estável). Além disso, possui duas funções: uma para fornecer o Global quando solicitado no início de uma transação de leitura-escrita e outra para fornecer o Estável para uma transação de somente leitura. Paralelo a isso, de forma assíncrona é executada uma função que verifica se uma transação de leitura-escrita foi efetivada. Caso positivo, o *timestamp* fornecido a ela passa a ser o novo TS (*timestamp*) Estável. Essa dualidade existe para que transações de somente leitura leiam dados consistentes e não aqueles que ainda estão sendo alterados por uma outra transação.

Na Figura 5 é exemplificado o modo como o protocolo MVCC organiza a execução de quatro transações hipotéticas ($TX_1$ a $TX_4$). Além disso, é ilustrado o estado do armazenamento ao longo do tempo. Neste modelo abordado, sempre que uma transação que realiza escrita iniciar, irá solicitar um TS Global, mesmo que ela não seja conflitante com outra (caso das transações $TX_1$ e $TX_2$). Mesmo a transação $TX_2$ sendo concluída antes, seu *timestamp* não é efetivado no TS Estável, pois isso poderia gerar inconsistência nos dados caso as transações alterassem uma mesma tupla. Isso interfere na execução da transação $TX_4$ que, por iniciar antes da conclusão da transação $TX_1$ e por ser uma transação de somente leitura, recebe um TS Estável que não reflete as alterações de nenhuma das transações iniciadas anteriormente, ao mesmo tempo em que não é travada pela execução de alguma outra transação. Quanto ao armazenamento, a cada nova escrita uma versão da tupla em questão é gravada com o *timestamp* da transação que realizou essa escrita.

**Figura 5 – Fluxo de execução do MVCC**




**Fonte: Traduzido de Wei *et al.* (2021)**

Por fim, é importante destacar que a proposta do protocolo MVCC não é exatamente nova. Bernstein e Goodman (1983) já realizavam um estudo sobre algoritmos aplicando esse protocolo ainda na década de 80. Porém, a demanda por sistemas distribuídos era baixa, assim como não havia tecnologia capaz de extrair todo o potencial do algoritmo. A novidade é a efetiva aplicação desse protocolo e sua melhoria continua para gerenciar a concorrência em SGBDs NewSQL. Quanto a isso, como destacado por Wei *et al.* (2021), mesmo que a aplicação do protocolo MVCC aumente o *throughput* do SGBD, o potencial ainda é limitado devido a necessidade da comunicação contínua com o oráculo. Por isso, existem variações na implementação e particularidades de cada SGBD, visando aumentar esse potencial de desempenho. Uma variação comum nos produtos NewSQL é combinar o uso do protocolo MVCC ao protocolo *two phase locking*, que será abordado na subseção 2.3.3.

### 2.3.3 Two phase locking

O protocolo *two phase locking* (2PL) é uma solução de controle de concorrência serializável para banco de dados distribuído (HARDING *et al.*, 2017). As transações executam em duas fases: fase de crescimento e fase de encolhimento (HARDING *et al.*, 2017). Na fase de crescimento, a transação obtem o *lock* das tuplas que irá acessar. O mesmo pode ser compartilhado ou exclusivo, sendo que o primeiro é fornecido para transações do tipo somente leitura e o segundo é fornecido para transações que realizam leitura e escrita. A diferença principal entre os modos de *lock* é que mais de uma transação pode ter o *lock* compartilhado de uma tupla, ou seja, transações simultâneas de somente leitura podem acessar o dado concorrentemente sem necessidade de uma delas esperar. Já no caso de *lock* exclusivo, apenas uma transação pode ter acesso a essa tupla por vez. Outras transações que tentem solicitar qualquer um dos tipos de *lock* para essa mesma tupla terão que aguardar em uma fila de espera. As estruturas de dados referentes as tabelas de *lock* individuais de tuplas (que geralmente contém também a fila de espera das transações que pretendem obter o *lock* dessa tupla), a tabela de *locks* de uma transação em específico e a lista de detecção de possíveis *deadlocks* são armazenadas e gerenciadas por cada nó do *cluster* isoladamente (BARTHELS *et al.*, 2019).

A fase de encolhimento é iniciada quando uma transação libera um dos *locks* que ela adquiriu. Uma vez nessa fase, a transação não poderá mais adquirir novos *locks*, porém pode continuar a realizar leituras e escritas nas tuplas que ainda possui o *lock*.

Nas implementações básicas do protocolo 2PL geralmente são utilizados dois mecanismos para evitar que uma transação entre em *deadlock*. O primeiro mecanismo é chamado *No Wait*, no qual não existe uma fila de espera para obter o *lock* das tuplas. Nesse mecanismo, se uma transação não pode obter o acesso imediatamente ela é então abortada e reiniciada (BARTHELS *et al.*, 2019). Já no mecanismo *Wait Die* são utilizados *timestamps* atribuídos no início das transações para ordená-las. Com essa informação, o SGBD coloca na fila de espera (para obter o *lock* da tupla) apenas as transações que tiverem um *timestamp* mais antigo que a

transação que tem o *lock* da tupla atualmente, as demais transações são abortadas (HARDING *et al.*, 2017). Utilizando algum desses mecanismos, o SGBD é capaz de evitar que transações entrem em *deadlocks*, ao custo de correr o risco de abortar transações de forma excessiva.

Os SGBDs NewSQL evitam utilizar somente o protocolo 2PL como sistema de controle de concorrência (PAVLO; ASLETT, 2016) por causa dos problemas com *deadlocks*. Porém, é comum que os SGBDs dessa categoria utilizem o 2PL com um MVCC aplicado a partir do momento que uma tupla é efetivada. Isso faz com que transações de somente leitura não precisem passar pela fase de adquirir um *lock* podendo ler diretamente o valor da tupla mais recente, reduzindo consideravelmente as situações de *deadlock*.

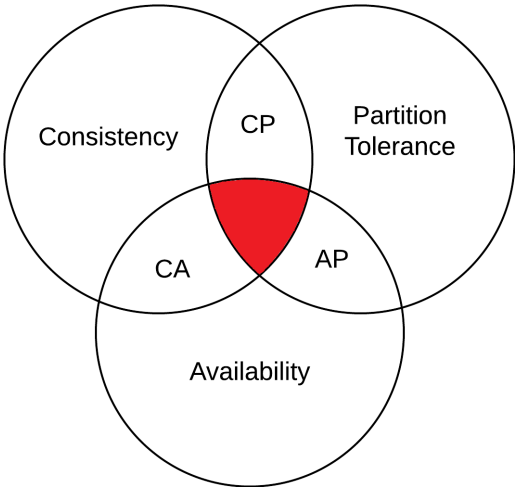
Como foi abordado, os SGBDs NewSQL não apresentam novas tecnologias, mas sim aplicam e aprimoram tecnologias já conhecidas, mas que não eram utilizadas junto ao modelo de armazenamento relacional devido à limitação de hardware e de conexão. Sendo a combinação de modelo relacional com sistema distribuído algo ainda não tão comum, é importante citar e abordar o comportamento de temas importantes relacionados à sistemas distribuídos e SGBDs NewSQL, como é o caso do teorema CAP detalhado na subseção 2.3.4.

### 2.3.4 Teorema CAP

O processamento das transações e a garantia das propriedades ACID sobre elas por parte do SGBD NewSQL, o qual opera em um sistema distribuído, tem relação direta com o teorema CAP (*Consistency*, *Availability*, *Partition tolerance*). Esse teorema idealizado por Brewer (2000) delimita o comportamento de aplicações distribuídas, o que incluí os Sistemas Gerenciadores de Bancos de Dados Distribuídos (SGBDDs). Grolinger *et al.* (2013) definem os elementos desse acrônimo: *Consistency* é equivalente a ter uma única instância atualizada de um dado, portanto todo o usuário obtém um mesmo dado caso o requisitem em um mesmo momento. O elemento *Availability* define que um dado deve estar disponível para ser usado por uma transação quando esta precisar dele. Já o elemento *Partition tolerance* define a capacidade do SGBD em continuar operando mesmo que a comunicação entre alguns nós seja perdida. Segundo o que foi proposto pro Brewer (2000), apenas dois elementos do acrônimo podem ser combinados ao mesmo tempo. A Figura 6 ilustra, usando um diagrama de Venn, as possibilidades de combinação que um SGBDD pode ter.

É possível realizar uma análise sobre o comportamentos dos SGBDs Relacionais, NoSQL e NewSQL quanto ao teorema CAP e como suas delimitações impostas impactam o funcionamento das aplicações. Para Kumar e Charu (2014) os SGBDRs estão contidos no conjunto CA, pois essa tecnologia não é trivialmente aplicada quando há perda de comunicação ou particionamento entre os nós (*Partition tolerance*) pois gera inconsistência de dados entre os componentes do *cluster*. Portanto não inclui o elemento *Partition tolerance* e atende assim ao teorema CAP. Também é possível afirmar que os SGBDRs atendem às propriedades *Consis-*

**Figura 6 – Diagrama de Venn do teorema CAP**




**Fonte: Autoria própria (2023)**

*tency* e *Availability*, pois até certo ponto conseguem se manter operacionais mesmo em caso de falhas.

Sobre as soluções NoSQL, Tailor, Choudhary e Jain (2014) destacam que inicialmente o teorema CAP era usado como uma justificativa para as aplicações apresentarem consistência eventual nos dados. Como esses SGBDs surgiram em um cenário no qual o escopo para crescimento no mercado era a necessidade de *throughput*, a escolha evidente foi abrir mão do elemento *Consistency* para garantir *Availability* e *Partition tolerance*, pois essas propriedades tem relação direta com o aumento da vazão de dados. Porém, algumas soluções NoSQL podem operar em modos distintos do teorema CAP, como é o caso do SGBD Cassandra, o qual suporta dois modos: modo AP que prioriza disponibilidade e tolerância a particionamento ou CA que prioriza disponibilidade e consistência (FEATHERSTON, 2010).

Quanto às soluções NewSQL, Tailor, Choudhary e Jain (2014) afirmam que "realmente quebram o teorema CAP. No NewSQL todas as três coisas são satisfeitas, mas o problema da latência ocorre ocasionalmente", o que é uma afirmação forte, pois implica que o SGBD é capaz de estar sempre disponível e operando com consistência. Já Ismail *et al.* (2021) afirmam que mesmo as soluções NewSQL precisam obedecer o teorema CAP e escolher um *trade-off* em caso de falha de comunicação entre os nós, pois isso impede que os protocolos sejam corretamente executados e que os dados sejam efetivados em todas as réplicas, gerando assim inconsistência. Segundo ele, o SGBD MemSQL (agora chamado SingleStoreDB) opera de forma semelhante ao SGBD Cassandra: oferece a possibilidade do DBA escolher qual combinação do teorema CAP seguir em caso de falhas que impossibilitem a comunicação entre nós do *cluster*, podendo operar tanto em CP quanto em AP. Já o SGBD VoltDB (2022) teve sua infraestrutura definida para manter os dados consistentes acima de tudo. Portanto, em caso de falha de um nó, o SGBD bloqueia-o de receber novas requisições. Quando esse nó retorna ao estado operacional e com plena capacidade de se comunicar, seus dados são atualizados de forma a

serem consistentes com as outras réplicas, para então voltar a atender requisições, atendendo portanto às propriedades CA quando submetido a falhas.

## 2.4 Discussões

No Capítulo 2 foi detalhado o modelo relacional e os modelos para o armazenamento de dados da classe NoSQL, os quais deram origem e embasamento para o surgimento da categoria NewSQL. Esta, agrupa as soluções relacionais que possuem uma arquitetura capaz de operar em um ambiente distribuído. Além disso, foram apresentados vários aspectos que compõe as soluções NewSQL e suas características marcantes. Com isso, será possível escolher e analisar uma solução dessa categoria quanto às características que foram definidas.

Chereja *et al.* (2021) em seu trabalho, além de apontar algumas características da categoria NewSQL, também relata sobre oportunidades de trabalhos futuros para a área de banco de dados, especificamente quanto à categoria NewSQL. Uma das sugestões é definir quais os critérios para uma solução fazer parte da categoria NewSQL, pois os utilizados atualmente são vagos e baseados na proposta inicial de Pavlo e Aslett (2016).

No Capítulo 3 será detalhado o processo de escolha da solução na qual serão aplicados os experimentos para validação das características do Quadro 1. Além disso, a arquitetura da solução escolhida será detalhadamente abordada, assim como o funcionamento do *benchmark* escolhido para realização dos experimentos.

# 3 MATERIAIS

Este capítulo apresenta os critérios utilizados para escolha do SGBD NewSQL a ser analisado quanto ao cumprimento das características definidas como critério para pertencer a categoria NewSQL elencadas no Quadro 1. Também descreve a arquitetura da solução escolhida, o ambiente computacional no qual foi aplicada e o *benchmark* ao qual a solução foi submetida. Primeiramente, a seção 3.1 descreve os critérios para escolher a solução de armazenamento NewSQL a ser avaliada. Já a seção 3.2 detalha a arquitetura do Google Cloud Spanner (2023c), solução escolhida para realização dos experimentos, destacando aspectos como o controle de concorrência e replicação dos dados. Ainda na seção 3.2 são apresentadas as técnicas aplicadas para fornecer consistência no ambiente distribuído, como a utilização de grupos Paxos juntamente a API (*Application Programming Interface*) *TrueTime* (CORBETT *et al.*, 2013). Por fim, a seção 3.3 descreve a ferramenta de *benchmark* utilizada nos experimentos.

## 3.1 Escolha da solução

Antes de detalhar a arquitetura da solução que será analisada e testada quanto às características definidas no Quadro 1, é necessário descrever os critérios utilizados para sua escolha. O Quadro 2 apresenta todas as informações elencadas e utilizadas no processo de escolha. A primeira dificuldade encontrada foi definir um grupo de SGBDs candidatos à serem material da pesquisa, pois não há um filtro no DB-Engines (2022) ou outras ferramentas de listagem de SGBDs para agrupar a categoria NewSQL. Portanto, a lista de candidatos foi definida com base nos trabalhos referenciados no Capítulo 2. Os critérios empregados no Quadro 2 e a influência dos mesmos na seleção da solução NewSQL são discutidos a seguir:

- **Número de citações**: A frequência com que um SGBD foi associado à categoria NewSQL nos trabalhos referenciados foi uma informação fundamental como critério de escolha, pois indica o quanto a solução em questão representa um consenso entre os autores sobre pertencer à categoria NewSQL. O limiar para aplicação desse critério foi a solução ser citada em pelo menos seis trabalhos. Com isso, considerou-se como candidatos os SGBDs: VoltDB com 13 citações; Google Cloud Spanner com 7 citações; Singlestore com 9 citações; NuoDB com 11 citações e CockroachDB com 6 citações.
- **Última atualização**: A data da última atualização que a solução recebeu pode ser um indicativo de como está o engajamento da equipe responsável com relação a agregar valor à solução em questão. A maioria dos SGBDs presentes no Quadro 2 foram atualizados recentemente. Uma exceção notável é o VoltDB, o qual não teve nenhuma atualização ao longo do último ano. Além disso, o NuoDB chama a atenção de forma desfavorável, uma vez que nem mesmo é possível localizar a data da última atualização em sua documentação. Esse critério foi utilizado para apoiar a escolha mas não foi suficiente para descartar uma solução , pois a equipe do VoltDB pode ter optado por

adotar atualizações anuais. Já a equipe do NuoDB pode ter optado por não manter as notas de atualizações disponíveis ao público.

- **Ambiente**: O ambiente (local ou em cloud) no qual o SGBD está disponível também é um critério relevante, principalmente porque indica quanto recurso é necessário investir para ter uma correta avaliação da arquitetura da solução. Optar por executar localmente implica no investimento em *hardware* para construção de um ambiente distribuído. Logo para avaliar o funcionamento em um sistema separado por distâncias geográficas seria necessário investir em mais de um *cluster*. Já o Google Cloud Spanner fornece a solução pronta para uso global, por exemplo. O ambiente escolhido para realização dos experimentos foi o *cloud* e por essa razão, o SGBD VoltDB foi descartado.
- **Período de teste**: Esse critério é especialmente relevante para comparar soluções em *Cloud*, pois elas possuem um custo de execução associado. Logo, é importante conhecer as possíveis políticas de teste. O Spanner, CockroachDB e Singlestore possuem métodos semelhantes para oferecer um período de teste. O Google Cloud Spanner e o CockroachDB oferecem 10GB (Gigabyte) de armazenamento para serem utilizados durante o período. Tanto o Spanner quanto o Singlestore fornecem uma quantidade de créditos para serem utilizados na plataforma, sendo U$400 e U$500 respectivamente. A forma como o crédito é descontado será abordado no critério "Custo da licença e/ou operação". Como não foi possível encontrar informações sobre cobranças do NuoDB (além disso, nenhuma informação sobre data de lançamento da última versão), o mesmo foi descartado.
- **Lançamento**: Outra informação utilizada como critério de escolha da solução foi sua data de lançamento. O objetivo é analisar uma solução nova no mercado, portanto a idade do projeto de arquitetura deve ser considerada. Dentre todas as soluções listadas no Quadro 2, o Google Cloud Spanner é o SGBD lançado ao público mais recentemente, em 2017. Portanto, é possível que sua arquitetura implemente tecnologias mais novas que seus concorrentes. Porém, isso não indica necessariamente que as outras soluções não implementam tecnologias novas em seus projetos, pois não há grande disparidade entre as datas de lançamento (4 anos entre o lançamento do Singlestore e do Spanner). Por isso esse critério não foi usado para descartar nenhuma solução, mas sim para embasar a escolha.
- **Posição no DB-engines**: Apesar do DB-Engines (2022) não possuir um filtro relacionado à categoria NewSQL, as soluções analisadas no Quadro 2 estão presentes na filtragem de SGBDs relacionais. Portanto a posição das soluções dentro do *ranking* dessa classe também é importante, dada a relevância do DB-Engines (2022) como uma referência sobre soluções de bancos de dados. Porém as opções que restaram (Google Cloud Spanner, CockroachDB e Singlestore) não apresentam grande disparidade de posição e, como a categoria NewSQL como um todo ainda não possui con-

senso da comunidade de desenvolvedores, essa posição pode não refletir a proposta e as características da arquitetura da categoria NewSQL. Por isso, esse critério não foi suficiente para descartar nenhuma das soluções, foi utilizado para embasar a escolha de uma delas.

- **Licença**: Conhecer o tipo de licença das soluções é importante, pois uma solução *open source* naturalmente não possui custo e tem todo o código disponível para o público. Já uma com licença comercial não disponibiliza o código para o público e, geralmente, possui custos para obtenção da licença. Dentre as opções, o CockroachDB possui licença *open source* (mesmo assim não é uma solução gratuita, pois possui o custo de execução em Cloud). Já o Spanner e o Singlestore possuem licença comercial. Apesar do CockroachDB ser a única *open source*, o fato de ter custo de execução faz com que esse critério não seja suficiente para que esta solução seja escolhida.
- **Custo da licença e/ou operação**: Apesar da maioria das soluções de armazenamento possuírem políticas de teste com tempos e capacidade de serviço variados, a possibilidade de não haver tempo suficiente para a realização dos experimentos foi considerada. Portanto o custo adicional de operação foi mais uma variável no processo de escolha. Como destacado no critério "Período de teste", tanto o Spanner quanto o Singlestore funcionam através de créditos. Sendo que a forma como esse crédito é descontado da conta também é semelhante entre eles. O primeiro desconta U$0.23 para cada GB utilizado cada mês e U$0.65 por hora de alocação da unidade de processamento. Já o Spanner desconta U$0.30 para cada GB utilizando durante cada mês e U$0.90 por hora de alocação de cada nó, sendo que um nó pode ter mais de uma unidade de processamento. O CockroachDB por sua vez não possui licença e pode ser contratado através de planos variados. Este não é um critério utilizado para descartar uma solução, mas sim para embasar uma escolha. Afinal, o custo adicional que os experimentos podem gerar é uma informação importante de ser considerada.

Os critérios definidos não foram suficientes para determinar a solução escolhida, pois não há um fator notável entre o Google Cloud Spanner, CockroachDB e o Singlestore que faça uma das soluções sobresair-se em relação as demais. Portanto, a ferramenta de *benchmark* utilizada na realização dos experimentos tornou-se um critério determinante. Durante a escolha da solução foi analisado que o YCSB (*Yahoo! Cloud Serving Benchmark*) poderia ser esta ferramenta, pois é capaz de gerar cargas de trabalho de curta duração com suporte a linguagem SQL, enquanto pode avaliar o desempenho de um sistema distribuído, aproximando-se das características definidas no Quadro 1. O fato do Google Cloud Spanner ser o único SGBD dentre as opções nativamente suportado pelo YCSB fez com que essa solução fosse a escolhida. Portanto, a arquitetura do Google Cloud Spanner será detalhada na seção 3.2. Posteriormente, o *benchmark* YCSB será detalhado na seção 3.3.

## Quadro 2 – Critérios para escolha da solução NewSQL

| Solução | Citado por | Nº de citações | Lançamento | Ambiente | Posição DB-engines | Licença | Período de teste | Última atualização | Custo da licença e/ou operação |
|---|---|---|---|---|---|---|---|---|---|
| VoltDB | Aslett (2021), Chaudhry(2020), Chereja (2021), Grolinger (2013), Harisson (2015), Ismail (2021), Khasawneh(2020), Knob (2019), Kumar (2014), Pavlo (2016), Schreiner (2019), Sivakumaran (2017), Tailor (2014) | 13 | 2010 | Local | 68 | Open Source | Não se aplica | Abril de 2022 | Não se aplica |
| Google Cloud Spanner | Aslett (2021), Chaudhry(2020), Chereja (2021), Grolinger (2013), Ismail (2021), Pavlo (2016), Sivakumaran (2017) | 7 | 2017 | Cloud | 47 | Comercial | 90 dias de teste, com acesso a 10Gb de armazenamento e U$400 de crédito para gastar na plataforma | Maio de 2023 | Sem licença, com custo de operação: U$0.90 por nó/hora U$0.30 GB/mês de armazenamento |
| Singlestore (MemSQL) | Aslett (2021), Chaudhry(2020), Chereja (2021), Ismail (2021), Knob (2019), Kumar (2014), Pavlo (2016), Schreiner (2019), Sivakumaran (2017) | 9 | 2013 | Cloud/Local | 36 | Comercial | U$500 para testar a plataforma, sem período | Maio de 2023 | Sem licença, com custo de operação: U$0.65 por hora U$0.23 GB/mês de armazenamento médio |
| Amazon Aurora | Aslett (2021), Ismail (2021), Pavlo (2016), Sivakumaran (2017) | 4 | 2015 | Cloud | 31 | Comercial | Sem período de teste | Não encontrei | Sem licença, com custo de operação: U$0.06 por hora da ACU |
| Azure SQL | Aslett (2021) | 1 | 2010 | Cloud | 10 | Comercial | U$200 de crédito para usar, sem período | Maio de 2023 | Sem licença, com custo de operação: U$0.52 vCorehora U$0.12 GB/mês de armazenamento |
| NuoDB | Chaudhry(2020), Chereja (2021), Grolinger (2013), Harisson (2015), Ismail (2021), Khasawneh(2020), Knob (2019), Kumar (2014), Pavlo (2016), Schreiner (2019), Tailor (2014) | 11 | 2013 | Cloud/Local | 87 | Comercial | Sem informações no site | Não encontrei | Não encontrei qualquer referência a valores |
| CockroachDB | Chaudhry(2020), Chereja (2021), Ismail (2021), Knob (2019), Pavlo (2016), Schreiner (2019) | 6 | 2015 | Cloud/Local | 35 | Open source | Sem restrição, 10Gb de armazenamento e 50 milhões de Request Units gratuitos cada mês no Cloud | Maio de 2023 | Sem licença |
| SAP HANA | Chereja (2021), Harisson (2015), Ismail (2021), Pavlo (2016) | 4 | 2010 | Cloud/Local | 16 | Comercial | Teste gratuito por 3 meses | Abril de 2023 | Sem licença, com custo de operação: EUR0.75 por unidade de processamento/mês |
| Citus | Chereja (2021) | 1 | 2010 | Cloud/Local | 58 | Open source | Não se aplica | Maio de 2023 | Não se aplica |
| Vitess | Chereja (2021) | 1 | 2013 | Cloud/Local | 103 | Open source | Não se aplica | Maio de 2023 | Não se aplica |
| FaunaDB | Chereja (2021) | 1 | 2014 | Cloud/Local | 78 | Comercial | Não tem | Junho de 2020 | U$25 mês de licença |
| HarperDB | Chereja (2021) | 1 | 2017 | Cloud/Local | 42 | Comercial | 0.5GB de computação e 1 GB de armazenamento grátis | Abril de 2023 | Sem licença, custos variáveis de acordo com o plano |
| MariaDB MaxScale | Chaudhry(2020), Ismail (2021), Pavlo (2016) | 3 | 2009 | Local | 9 | Open source | Não se aplica | Março de 2023 | Não se aplica |

OBS: Dados obtidos em Maio de 2023

**Fonte: Autoria própria (2023)**

## 3.2 Google Cloud Spanner

A história da computação em nuvem remonta a década de 50, quando surgiram desafios econômicos e práticos relacionados à disponibilização de um *mainframe* para cada funcionário de uma empresa. O que acabou gerando a ideia de criar uma forma do funcionário poder acessar um mainframe central de forma remota. Em 1970, o conceito de virtualização avançou com a introdução do software VMware, permitindo a execução simultânea de vários sistemas operacionais em um único ambiente físico (ARIF *et al.*, 2019).

Diversos avanços ocorreram nas décadas posteriores, culminando no lançamento de várias tecnologias em *cloud* na década de 2000, com destaque para o Google Docs em 2006, familiarizando o público com o conceito de armazenamento em nuvem. No mesmo ano, a Amazon introduziu o EC2, permitindo que usuários individuais e pequenas empresas alugassem computadores para executar suas próprias aplicações. No final da década de 2000, a computação em nuvem cresceu significativamente, com a chegada de mais aplicativos baseados em navegador provisionados pela Google e por outras empresas (ARIF *et al.*, 2019).

A computação em nuvem permitiu que indivíduos e organizações acessem os recursos de computação sob demanda, de qualquer lugar e em qualquer momento. Além disso, o surgimento de empresas com uma vasta quantidade de serviços em nuvem, como a Google, possibilitou que os custos e complicações que existem em desenvolver os sistemas próprios (como necessidade de alocar uma equipe e a própria manutenção do sistema) pudessem ser centralizados nos serviços oferecidos por essas empresas (SUNYAEV; SUNYAEV, 2020). As ferramentas e aplicações que podem ser acessadas por meio da nuvem vão desde *Software as a Service (SaaS)* como o editor de texto Google Docs, até máquinas virtuais e SGBDs instalados neste ambiente conhecidos por *Database as a Service (DBaaS)*, como é o caso do Google Cloud Spanner.

Anunciado em 2013 pela Google como uma solução exclusivamente interna, o Google Cloud Spanner é um SGBD capaz de facilitar o processo de integração das aplicações com os seus respectivos dados. Na ocasião, Corbett *et al.* (2013) destacaram que o projeto do Spanner surgiu da dificuldade em que alguns desenvolvedores tinham para integrar as aplicações ao *BigTable*, solução de banco de dados essa que operava na maioria dos serviços da Google até então. As principais dificuldades descritas estavam relacionadas à necessidade de escrever códigos excessivamente complexos para processar e agregar os dados oriundos do armazenamento chave-valor, visto que esses não possuem um esquema definido e também não possuem uma linguagem de consulta estruturada que possibilite a agregação dos dados. Além disso, havia um trabalho extra para assegurar a consistência dos dados em alguns casos nos quais as aplicações necessitavam dessa característica. Neste contexto, o Spanner foi desenvolvido para cumprir o desafio de satisfazer essas necessidades sem apresentar grande perca de desempenho em relação ao *BigTable*.

O Spanner permaneceu apenas como um recurso interno por um longo período, sendo utilizado por algumas aplicações, como o *AdWords* e o Google Play por exemplo (SABHARWAL; EDWARD, 2019). Somente em 2017 foi anunciado que o Spanner se tornaria disponível para todo o público através do Google Cloud Platform, se juntando a outros recursos semelhantes disponíveis nela, como o BigTable e o Google Cloud Storage. Sua proposta é ser um SGBD altamente disponível, compatível com a linguagem SQL, horizontalmente escalável, multi-versão, globalmente distribuído e replicado de forma síncrona, além de fornecer consistência-externa nas transações distribuídas. Essas características são oferecidas através de uma arquitetura que une tecnologias já consolidas, como o uso do algoritmo de consenso paxos para coordenar transações de forma que as réplicas assumam dados consistentes, juntamente à novas tecnologias, como a API *TrueTime* desenvolvida especialmente para o Spanner. A implementação dessa API possibilita o multi-versionamento de tuplas e a atribuição de *timestamps* à transações de forma global. As tecnologias do algoritmo de paxos e a API *TrueTime* serão abordadas na subseção 3.2.2 e subseção 3.2.4. Outras particularidades da arquitetura desse SGBD serão detalhadas na presente seção

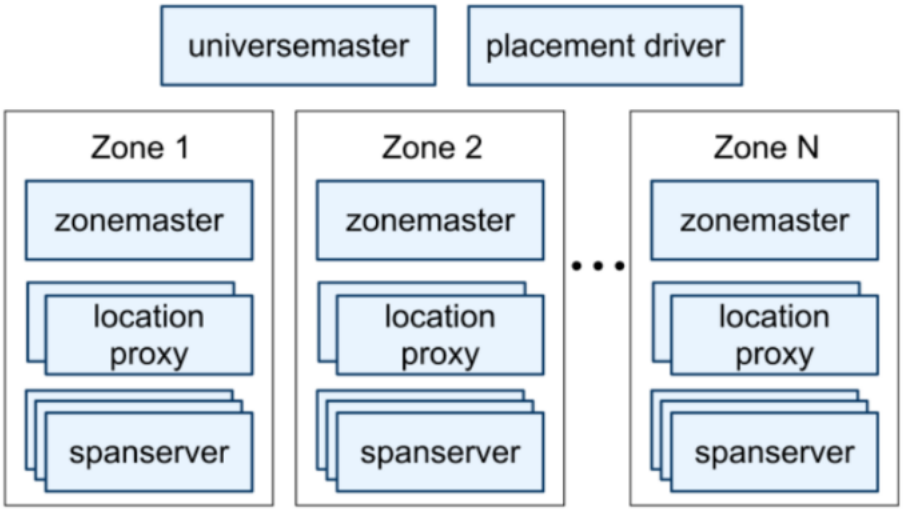
### 3.2.1 Ambiente distribuído

O conjunto de todos os *datacenters* ao redor do mundo que executam uma dada versão do Spanner é chamado de *universe*. Há apenas um *universe* no ambiente de produção, o qual é dividido em um conjunto de *zones* pelos continentes. *Zones* são unidades administrativas que podem ser adicionadas e removidas de forma dinâmica ao *universe* enquanto o sistema está em execução. As *zones* possuem autonomia para gerenciar seus dados, tendo uma máquina denominada *zonemaster* responsável por organizar as operações que serão executadas nos vários *spanservers*, os quais efetivamente armazenam e manipulam os dados.

Os clientes que acessam o Spanner localizam seus dados nos diversos *spanservers* através do *location proxy* de cada *zone*, que funciona como uma mapeador para o *zonemaster* direcionar os clientes aos *spanservers* corretos. O *universe* possui ainda um componente único chamado *placement driver*, o qual é responsável por mover dados entre as diversas *zones* e outro componente chamado *universemaster* que oferece um terminal para acesso de informações analíticas sobre as *zones* (MO, 2022). A Figura 7 ilustra os elementos descritos e como eles estão organizados em um *universe* do Spanner.

As *zones* específicas que um usuário deseja utilizar são indicadas no processo de criação da instância, sendo que uma instância do Spanner é "praticamente análoga a um servidor de um SGBD Relacional" (SABHARWAL; EDWARD, 2019). Ainda segundo Sabharwal e Edward (2019), a instância "contém uma ou mais bases de dados e tem recursos alocados (de hardware e armazenamento) que são utilizados por todas as suas bases de dados". Ou seja, na criação da instância, o usuário define os parâmetros técnicos que deseja utilizar na respectiva instância do Spanner, como o número de unidades de processamento que deseja ter disponível

**Figura 7 – Estrutura do ambiente Spanner**




**Fonte: (CORBETT *et al.*, 2013)**

em sua solução e a quantidade de réplicas que serão criadas, assim como em quais *zones* ficarão armazenadas. Cada 1000 unidades de processamento equivalem a um nó, porém uma instância pode ser definida com menos unidades, o que significa que o Spanner não reserva uma máquina completa para esta instância.

Diferentes *zones* estão estabelecidas em diferentes localizações, portanto o Spanner possui um algoritmo em sua arquitetura para realizar e controlar a replicação dos dados entre as *zones*. O núcleo desse algoritmo consiste na divisão da base de dados em diferentes grupos paxos, conceito este que será abordado na subseção 3.2.2.

Por executar em um ambiente distribuído, o Google Cloud Spanner está sujeito às delimitações impostas pelo teorema CAP que foram apresentadas na subseção 2.3.4. Sobre esse aspecto, o propósito da Google com o Google Cloud Spanner é oferecer um SGBD consistente e altamente disponível, embora não seja possível afirmar de forma concisa que não existem partições ocasionadas por problemas de comunicação entre os nós. Elas podem ocorrer, mesmo que com pouca frequência, implicando no Spanner estar efetivamente no grupo CP do teorema CAP, priorizando sempre manter a consistência em caso de falhas, cedendo o espectro da disponibilidade. Porém, na prática, os desenvolvedores e DBAs tem a impressão de que o Spanner está no grupo CA do teorema CAP, pois a disponibilidade é tão alta ao ponto de que podem ignorar o tratamento de erros por indisponibilidade, chegando a ser disponível na ordem de 99,99999% do tempo segundo Brewer (2017).

Alguns aspectos contribuem para o Spanner ser altamente disponível, como o fato de utilizar uma própria rede privada da Google para comunicação entre os *datacenters*. Além disso, cada *datacenter* está conectado à rede por meio de pelo menos três cabos de fibras independentes, criando redundância e mitigando falhas causadas por problemas como corte do cabo.

Outra característica implementada que aumenta a disponibilidade é o fato de utilizar grupos paxos para efetivação das transações, assunto abordado na subseção 3.2.2. Uma vez que a maioria dos nós que compõem um grupo estejam conectados à rede, o sistema continua processando transações normalmente (BREWER, 2017).

### 3.2.2 Algoritmo paxos e replicação

O algoritmo paxos surgiu como uma abstração de um protocolo desenvolvido e aplicado séculos atrás na ilha de Paxos, na Grécia. Esse protocolo foi criado para que decretos de lei pudessem ser aprovados sem o aval de todos os legisladores que compunham o parlamento. Nesta época, a aprovação de novos decretos gerava dificuldades pois nem sempre todos os legisladores estavam presentes na câmara, o que poderia gerar atraso no processo e até mesmo inconsistência nas leis conhecidas por cada legislador (LAMPORT, 2019). A solução deste problema foi a aplicação de um protocolo composto, de forma resumida, das seguintes proposições:

- Cada legislador deve manter seu próprio livro de decretos, com decretos ordenados por um número relativo a quando foi proposto.
- Uma vez escrito no livro, o decreto não pode ser alterado.
- Qualquer legislador pode propor um novo decreto, o qual será registrado pelos legisladores caso não haja em seus respectivos livros um decreto com aquele número.
- Se a maioria dos legisladores estiver na Câmara e nenhum entrar ou sair dela por um período de tempo, então qualquer decreto proposto será aprovado e irá aparecer nos livros dos legisladores.
- A comunicação entre os legisladores é feita através de mensageiros. Portanto uma mensagem sobre um decreto pode chegar múltiplas vezes ou até mesmo nenhuma vez.

De forma resumida, o protocolo utilizado em Paxos consiste em seguir essas proposições para que um decreto seja considerado válido, o que ocorre quando for efetivado pela maioria dos legisladores. O decreto eventualmente será replicado para os demais legisladores pois há troca de informação entre eles. Além disso, qualquer decreto que seja proposto com o mesmo número de um decreto já presente no livro do legislador será negado.

Alguns séculos depois do protocolo ser aplicado na sociedade de Paxos, o contexto da computação distribuída deu espaço para que um algoritmo de consenso fosse proposto baseado nele, o qual foi inicialmente apresentado por Leslie (1998). O algoritmo em questão tem como objetivo garantir que um único valor seja escolhido dentre vários que podem ser propostos em um sistema que executa de forma distribuída, garantindo desta forma a consistência dos dados entre os componentes do sistema.

Em um algoritmo paxos existem três papéis que precisam ser cumpridos: *proponente*, *aceitador* e *aprendiz*. O primeiro é responsável por enviar uma proposta de novo valor ou alte-

ração de um valor existente aos *aceitadores*. O *aceitador* por sua vez tem apenas a função de comparar o valor recebido e responder se a proposta é válida. Já o *aprendiz* é acionado apenas no final do processo, para efetivar em seu sistema o valor em caso da proposta ser aceita pela maioria dos *aceitadores*. As etapas de execução do algoritmo são listadas na sequência:

- 1º: O *proponente* envia uma requisição de preparação com a proposta para um conjunto de *aceitadores*. As propostas são ordenadas numericamente, portanto o *proponente* deve associar à proposta um *n* maior do que o número associado à última proposta conhecida.
- 2º: O *aceitador* responde com uma promessa de não responder uma proposta com valor inferior a *n*. Caso tenha aceitado uma proposta de preparação para o valor em questão anteriormente com um número inferior a *n*, envia esta proposta junto a resposta.
- 3º: Quando a maioria dos *aceitadores* responderem a requisição de preparação, o *proponente* envia uma requisição de aceitação para toda a lista de *aceitadores*.
- 4º: Quando um *aceitador* recebe uma requisição de aceitação para uma proposta de valor *n*, o mesmo confirma a proposta a menos que já tenha respondido a uma requisição de preparação com uma proposta com valor maior que *n*.
- 5º: Os *aprendizes* replicam o novo valor que foi aceito, seja através de uma requisição enviada pelo *aceitador* ou por consulta a outros *aprendizes* (depende da implementação).

De forma resumida, seguindo essas etapas, o algoritmo de consenso de paxos é capaz de escolher um valor consistente para o sistema (LAMPORT, 2001). Algumas melhorias como a escolha de um *líder* distinto para realizar as propostas e um *aprendiz* distinto para realizar a replicação do valor aceito podem ser utilizadas para evitar que o algoritmo fique preso em *loopings*.

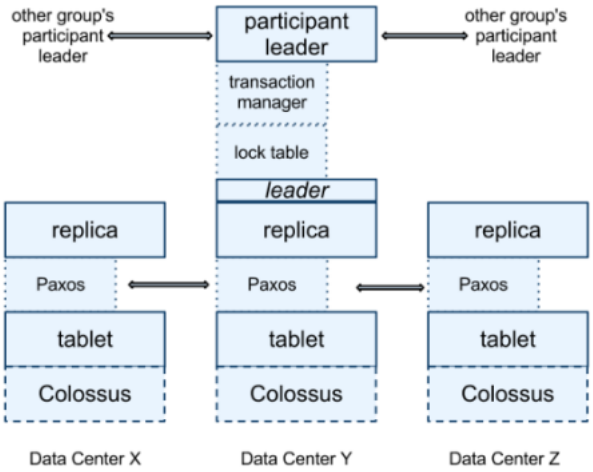
No contexto do Google Cloud Spanner, cada *spanserver* implementa uma máquina de estados paxos em cada réplica, como pode ser visto na Figura 8. O conjunto de réplicas é chamado de Grupo Paxos, neste cada réplica pode assumir qualquer um dos três papéis definidos no algoritmo paxos, inclusive com uma réplica podendo exercer os três papéis ao mesmo tempo. Além disso, o paxos implementado pelo Spanner tem escolha de líderes que podem permanecer no papel por um longo tempo (tanto o líder *proponente* quanto o *aprendiz*). Alguns elementos da Figura 8 serão abordados na subseção 3.2.3, como o caso do *tablet* e o *Colossus*. Já os componentes da parte superior, como o *transaction manager* e a *lock table* serão abordados na subseção 3.2.5.

Cada máquina de estados armazena seus próprios metadados e *logs* relacionados às operações (CORBETT *et al.*, 2013). Quando uma réplica que está desatualizada recupera conexão com o restante das réplicas, ela atualiza seus dados utilizando o *log* de uma que está atualizada. O conceito de máquinas de estados paxos é utilizado para alcançar uma replicação consistente dos dados.

A quantidade de réplicas e onde elas serão localizadas são informações configuráveis durante a criação de uma instância. O Spanner suporta replicação em várias *zones* espalhadas através do mundo. Porém, replicar dados em diferentes continentes naturalmente reduz o *throughput* do sistema, dada a qualidade da comunicação entre os *datacenters* e a distância entre os mesmo inevitavelmente serem fatores que afetam o desempenho. A implementação do algoritmo de consenso paxos busca reduzir essa perda de desempenho.

Operações de escrita submetidas ao Spanner desencadeiam o protocolo paxos envolvendo todas as réplicas que possuem o dado alterado. Já as operações de leitura podem ser realizadas diretamente na *tablet* (a estrutura será abordada na subseção 3.2.3) de qualquer réplica suficientemente atualizada, evitando assim acionar o protocolo (CORBETT *et al.*, 2013). Outros elementos da arquitetura como a replicação dos dados e as estruturas de dados presentes no Spanner tem relação direta com os grupos de paxos, sendo que este relacionamento será abordado na subseção 3.2.3.

**Figura 8 – Estrutura de um spanserver**




**Fonte: (CORBETT *et al.*, 2013)**

### 3.2.3 Modelagem dos dados

O Spanner, apesar de apresentar a linguagem SQL para consulta e manipulação dos dados, armazena os dados usando o um formato chave-valor semelhante ao utilizado no *BigTable* (CORBETT *et al.*, 2013). A principal diferença entre o dado armazenado no Spanner e no *BigTable* é a necessidade de armazenar um *timestamp* juntamente aos dados para permitir um uso do MVCC, fazendo com que no Spanner seja representado da seguinte forma:

Como ilustrado na Figura 8, cada réplica possui um elemento chamado *tablet*. Essa parte da réplica agrupa várias estruturas de dados *tablet*, sendo que cada uma dessas estruturas armazena vários dados do tipo chave-valor no formato representado no Quadro 3. O estado

**Quadro 3 – Formato chave-valor utilizado pelo Spanner**

| (key: String, timestamp: int64) -> String |
|---|

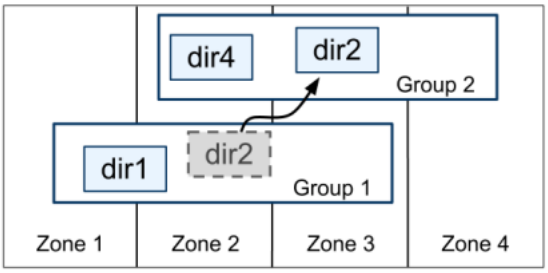
**Fonte: (CORBETT *et al.*, 2013)**

de cada *tablet* é armazenado em um sistema de arquivos distribuído chamado *Colossus* (MO, 2022).

Para otimizar a localidade dos dados e a replicação geográfica, o Spanner introduz um conceito de agrupamento chamado de *diretórios*. Os dados são agrupados nessas estruturas de forma que "contenham chaves contíguas que compartilham um prefixo" (MO, 2022). Isso permite que os aplicativos agrupem dados comuns próximos uns dos outros por meio da definição correta dessas chaves. Uma estrutura *tablet* pode conter vários diretórios, tornando também possível agrupar dados que não tem um intervalo de chaves contíguas (MO, 2022).

O processo de mover dados entre diferentes grupos paxos é feito *diretório* por *diretório*, processo este representado na Figura 9, na qual o diretório *dir2* é movido do grupo paxos denominado *Group 1* ao *Group 2*. A possibilidade de mover diretórios inter-paxos torna possível que o Spanner identifique *diretórios* que são frequentemente acessados juntos e os coloque em um mesmo grupo. Esse "transporte" de *diretórios* é feito através de uma tarefa executada em segundo plano chamada *Movedir*. A execução dessa tarefa não reduz a disponibilidade do SGBD, visto que o *diretório* não fica indisponível durante o processo (CORBETT *et al.*, 2013).

**Figura 9 – Comportamento dos diretórios em um grupo paxos**




**Fonte: (CORBETT *et al.*, 2013)**

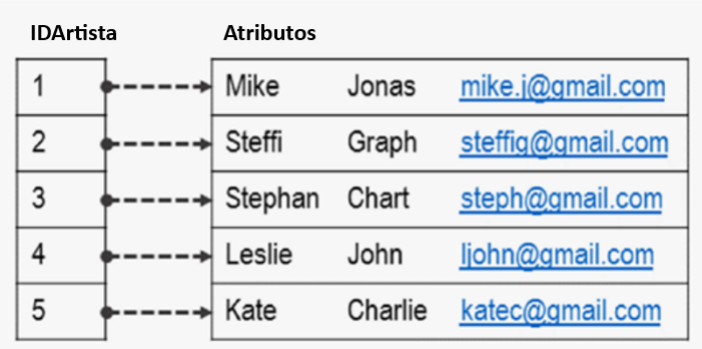
O Spanner oferece um modelo de dados baseado em tabelas semi-relacionais esquematizadas e transações de propósito geral (abordadas na subseção 3.2.5). Não é possível afirmar que o modelo de dados do Spanner é puramente relacional pois toda tupla precisa ter uma ou um conjunto de chaves primárias (CORBETT *et al.*, 2013). Essas chaves primárias "apontam" para o restante dos atributos de cada tabela, de forma semelhante ao que ocorre em um armazenamento chave-valor (MO, 2022). Um exemplo de como representar os dados de um artista em uma tabela do Spanner pode ser visto no Quadro 4 e como os dados inseridos nessa tabela são armazenados pode ser visto na Figura 10.

**Quadro 4 – Esquema da tabela Artista**

| Nome do atributo | Tipo de dado | Limitação |
|---|---|---|
| IDArtista | INT64 | Chave primária |
| PrimeiroNome | STRING{[}30{]} | |
| UltimoNome | STRING{[}20{]} | |
| Email | STRING{[}50{]} | |

**Fonte: Adaptado de Sabharwal e Edward (2019)**

**Figura 10 – Representação física dos dados**




**Fonte: Adaptado de Sabharwal e Edward (2019)**

O Spanner possui um recurso adicional para manipular relacionamentos entre tabelas que são frequentemente requisitadas juntas. Isso é feito por meio da possibilidade de definir relacionamentos hierárquicos entre elas, o que faz com que tuplas das tabelas sejam localizadas juntas no disco, melhorando a eficiência em obtê-las (SABHARWAL; EDWARD, 2019). O relacionamento é definido na criação da tabela filha utilizando a declaração INTERLEAVE IN PARENT, referenciando qual será a tabela "pai" no relacionamento. A Figura 11 exemplifica a criação da tabela *Album* sendo filha da tabela *Artist*. Como resultado, os dados da tabela *Artist* e da tabela *Album* ficaram entrelaçadas no armazenamento, o que é representado na Figura 12.

**Figura 11 – Criação de tabela com relacionamento**

```
CREATE TABLE Album (
  IDArtista INT64,
  IDAlbum INT64,
  TituloAlbum STRING(50),
) PRIMARY KEY (IDArtista, IDAlbum),
  INTERLEAVE IN PARENT Artist ON DELETE CASCADE;
```

**Fonte: Autoria própria (2023)**

### 3.2.4 TrueTime

A API TrueTime foi desenvolvida para proporcionar uma maneira robusta e confiável de tratar a representação e manipulação de tempo em um ambiente sincronizado globalmente. No Google Cloud Spanner é utilizada como uma ferramenta para obtenção de instantes de tempos em que as ações são executadas (como criação e alteração de tuplas). O principal

**Figura 12 – Entrelaçamento das tabelas Album e Artist**

| Chave | Atributos | | |
|---|---|---|---|
| Artists(1) | Mike | Jonas | mike.j@gmail.com |
| Albums(1,1) | Junk Yard | | |
| Albums(1,2) | Go On | | |
| Artists(2) | Steffi | Graph | steffig@gmail.com |
| Albums(2,3) | Girls Like You | | |
| Albums(2,4) | Colour Green | | |
| Albums(2,5) | Purple | | |
| Artists(3) | Stephan | Chart | steph@gmail.com |
| Artists(4) | Leslie | John | ljohn@gmail.com |
| Artists(5) | Kate | Charlie | katec@gmail.com |

**Fonte: Adaptado de Sabharwal e Edward (2019)**

diferencial dessa API é sua capacidade de contornar a incerteza no tempo que é produzida pela dificuldade de sincronizar com precisão relógios distintos. Essa incerteza é contornada com o conceito de representar o tempo usando intervalos e não instantes de tempo. Portanto, quando uma operação é realizada no Spanner é atribuído à ela um intervalo de tempo que contém o instante exato em que ela aconteceu e não o instante exato em si. Os intervalos de tempo surgem da adição do erro em um instante de tempo, tanto para mais como para menos. Permitindo que os clientes que consomem dados do Spanner tenham a segurança de que a margem de erro inserida no intervalo (CORBETT *et al.*, 2013) garante que dois *timestamps* não são sobrepostos. Os métodos presentes na API TrueTime estão representados no Quadro 5.

**Quadro 5 – Métodos da API TrueTime**

| **Método** | **Retorno** |
|---|---|
| TT.*now*() | TT*interval*: [maisAntigo, maisNovo] |
| TT.*after*(TT*stamp* t) | verdadeiro se *t* com certeza já passou do TT*interval* atual |
| TT.*before*(TT*stamp* t) | verdadeiro se *t* com certeza está antes do TT*interval* atual |

**Fonte: Adaptado de Corbett *et al.* (2013)**

O método central da API TrueTime é o TT.*now*(), que retorna um dado do tipo *TTinterval* que consiste em um intervalo que contém o tempo absoluto no momento da invocação. A API também fornece métodos de conveniência, como o TT.*after*(TT*stamp* t) e o TT.*before*(TT*stamp* t), que são envoltórios em torno do método TT.*now*(). Esses métodos auxiliam na comparação temporal entre eventos, facilitando determinar se um evento ocorreu antes ou depois de outro.

Para garantir a precisão e confiabilidade da API, o TrueTime utiliza como fontes de tempo relógios GPS e relógios atômicos de forma redundante. O primeiro consiste em dispositivos que combinam sinais recebidos de múltiplos satélites para reduzir erros e oferecer valores de tempo mais precisos. Já os relógios atômicos são dispositivos que fornecem medidas de tempo

com precisão elevada pois utilizam o princípio físico da mudança de camada do elétron em um átomo para determinar os instantes de tempo, sendo que a base para sua precisão é a constância com que essa transição de camadas ocorre em certos átomos. A escolha de duas fontes de referência distintas tem como objetivo reduzir falhas associadas a cada tipo de relógio. Os relógios GPS podem ser afetados por falhas na antena, interferência local de rádio e até mesmo interrupções no sistema GPS. Por outro lado, os relógios atômicos podem falhar de maneiras não correlacionadas com os relógios GPS, incluindo desvios de frequência ao longo do tempo (CORBETT *et al.*, 2013).

A implementação do TrueTime consiste na disposição de uma coleção de máquinas *time master* em cada *datacenter*, bem como um *timeslave daemon* por nó. As máquinas do tipo *time master* são responsável por implementar os métodos do Quadro 5, sendo que a maioria utiliza receptores GPS com antenas dedicadas e com separação física para reduzir efeitos de interferência. Uma quantidade menor de *time masters* são equipadas com relógios atômicos, essas são chamadas de *Armageddon masters*. Continuamente são feitas comparações entre as referências de tempo das diversas *time masters*, de modo a reduzir divergências e aumentar a acurácia.

O *timeslave daemon* em cada máquina consulta vários *time masters* antes de fornecerem o tempo para a operação que o solicitou. Essa abordagem colaborativa ajuda a reduzir a vulnerabilidade a erros de uma única fonte de referência. Desse modo, a API TrueTime é capaz de oferecer uma forma de abordar a incerteza temporal em sistemas distribuídos, proporcionando solidez ao sistema de multi-versionamento de tuplas implementado pelo Google Cloud Spanner.

### 3.2.5 Controle de concorrência e transações

Como destacado na subseção 3.2.2, o Spanner implementa um mecanismo para escolha de um líder responsável por gerenciar todas as propostas submetidas à aquele grupo. Além disso, a réplica do *spanserver* que está exercendo o papel de líder implementa uma *lock table* para o controle de concorrência. Ou seja, o Spanner utiliza o mecanismo de *two phase locking* descrito na subseção 2.3.3 para o gerenciar transações concorrentes dentro de um grupo paxos.

A réplica que exerce o papel de líder também implementa o *transaction manager* para suportar transações distribuídas. Caso uma transação de leitura/escrita seja satisfeita por somente um grupo paxos, ela pode ignorar os procedimentos do *transaction manager*, pois a *lock table* e o próprio grupo paxos já garantem a transacionalidade. Agora, caso a transação envolva mais de um grupo paxos, os líderes de cada grupo envolvidos na transação executam o protocolo *two phase commit*. Esse protocolo funciona da seguinte forma (HARDING *et al.*, 2017):

- Um dos líderes é escolhido para atuar como *coordenador*;

- O *coordenador* envia uma mensagem de preparo para todos os demais participantes;
- Durante a fase de *preparo*, os participantes votam se a transação deve ser efetivada ou abortada;
- Depois de enviar todas as mensagens, o *coordenador* entra na fase de *efetivação*;
- Caso algum dos *participantes* votar para abortar, o *coordenador* envia uma mensagem para todos os participantes abortarem a transação. Caso contrário, envia uma mensagem de efetivação.

Transações do tipo somente leitura podem ser executadas ignorando tanto a *lock table* quanto o *transaction manager*, pois não alteram os dados armazenados. A implementação desses mecanismos, juntamente a API *TrueTime* para fornecimento dos *timestamps* faz com que o Spanner seja capaz de fornecer consistência a seus dados. Tanto a *lock table* quanto o *transaction manager* estão representados na Figura 8, que ilustra a estrutura de um *spanserver* com um grupo paxos, sendo que o *Datacenter Y* possui a réplica que está atuando como líder *proponente*.

Além das transações de leitura/escrita e transações de somente leitura, o Spanner também fornece suporte a leitura de *snapshots*. Esse tipo de transação consiste na obtenção de tuplas em um dado momento do tempo (definido por meio de um *timestamp*) ou de tuplas mais novas do que um *timestamp* informado. Essa funcionalidade permite que sejam realizadas leituras no "passado", facilitando auditoria e recuperação de dados. Assim como transações de somente leitura, as leituras de *snapshots* não precisam passar pela *lock table* (CORBETT *et al.*, 2013).

As leituras podem ser realizadas em qualquer réplica que esteja "suficientemente atualizada". Cada réplica possui um valor de *timestamp* chamado $t_{safe}$ que define qual *timestamp* máximo a réplica garante estar atualizada. Portanto, uma réplica pode responder a uma leitura de *timestamp* $t$ se $t <= t_{safe}$. O valor de $t_{safe}$ é definido para cada réplica com base no valor mínimo entre o *timestamp* utilizado na última escrita submetida ao grupo paxos e o menor *timestamp* de uma transação que esteja em fase de preparação nesse grupo (CORBETT *et al.*, 2013).

As transações do Spanner seguem algumas invariantes para garantir a corretude do sistema, as quais estão listadas no Quadro 6. Essas invariantes delimitam o comportamento do sistema principalmente durante o controle de concorrência e o processo de associar *timestamps* às efetivações de transações. A invariante de *consistência externa* implica que duas transações que iniciem simultaneamente e manipulem os mesmos dados não podem ser executadas paralelamente. Ao invés disso, é aplicado o mecanismo de *wound-wait* no qual a primeira transação que obter o *lock* da tupla prossegue com sua execução enquanto a segunda é abortada e iniciada novamente somente quando o *lock* for liberado, evitando assim *deadlocks* no sistema (CORBETT *et al.*, 2013). A implementação desse mecanismo se faz necessária pois as escritas que ocorrem em uma transação são armazenadas no *buffer* e somente realizadas de fato após a efetivação, quando recebem um *timestamp*.

**Quadro 6 – Invariantes do Cloud Spanner**

| Invariante | Descrição |
|---|---|
| Disjunção | Em cada grupo paxos, o intervalo de concessão do cargo de líder é disjunto de todos os outros líderes. |
| Monotonicidade | O Spanner atribui *timestamps* as escritas que ocorrem em um grupo paxos em ordem crescente e de forma uniforme. |
| Consistência externa | Se uma transação *T2* ocorrer após a efetivação de uma transação *T1*, o *timestamp* de *T2* deve ser maior do que o de *T1*. |

**Fonte: Adaptado de (MO, 2022)**

## 3.3 Benchmark YCSB

O YCSB consiste em um *benchmark* capaz de gerar cargas de trabalho e analisar o desempenho de SGBDs quando submetidos a essas cargas. Foi inicialmente desenvolvido para ser aplicado em SGBDs disponíveis em Cloud que utilizam o modelo chave-valor para armazenamento. Porém, eventualmente também passou a suportar SGBDs configurados localmente como o VoltDB e o Postgres, além de outros modelos de armazenamento, com destaque ao modelo relacional (COOPER *et al.*, 2010).

O principal elemento do YCSB é o pacote com diferentes cargas de trabalho, chamado de *YCSB Core Package*. Esse pacote oferece uma coleção de *workloads* desenvolvidos para avaliar diferentes aspectos de um SGBD, principalmente relacionados à performance. Cada *workload* contém um conjunto de operações triviais do sistema, como leituras, atualizações e inserções. Segundo Cooper *et al.* (2010), o YCSB não foi desenvolvido para simular aplicações reais como é o caso de outros *benchmarks*, mas sim para avaliar uma gama de características por meio de operações simples e de dados genéricos utilizando diversas *threads* para simular conexão de clientes com o recurso de armazenamento de dados. Oferecendo uma forma distinta de avaliar o desempenho do SGBD.

Os dados gerados pelo YCSB são armazenados em uma tabela chamada *usertable*. Essa tabela possui um campo de chave primária chamado *id* e *n* campos de valores chamados de *field0*, *field1* e assim por diante. O valor desses campos consiste em um conjunto de caracteres gerados aleatoriamente, sem um significado associado. Já o campo de *id* é definido para cada registro no formato "*user*X", sendo *X* um valor numérico também aleatório. O número de campos em cada tabela pode ser definido com o parâmetro *fieldcount*, sendo que o padrão é 10 campos. A Figura 13 destaca a criação da tabela utilizada pelo YCSB no Spanner.

Os *workloads* do YCSB são definidos utilizando parâmetros que controlam as proporções com que serão realizadas leituras de tuplas únicas, atualizações, inserções, *scans* (leitura de uma série de tuplas em um intervalo de id's) e qual algoritmo de distribuição uniforme será utilizado durante a carga de trabalho. Os parâmetros apresentados a seguir são utilizados para diferenciar os *workloads* disponibilizados pelo *YCSB Core Package*. Além disso podem ser usadas combinações distintas deles para produzir novos *workloads*:

**Figura 13 – Criação da tabela utilizada pelo YCSB no Spanner**

```
CREATE TABLE usertable (
  id STRING(MAX),
  field0 STRING(MAX),
  field1 STRING(MAX),
  field2 STRING(MAX),
  field3 STRING(MAX),
  field4 STRING(MAX),
  field5 STRING(MAX),
  field6 STRING(MAX),
  field7 STRING(MAX),
  field8 STRING(MAX),
  field9 STRING(MAX),
) PRIMARY KEY(id);
```

**Fonte: Autoria própria (2023)**

- *readproportion* (de 0 a 1): Define a proporção de operações de leitura realizadas;
- *updateproportion* (de 0 a 1): Define a proporção de atualizações nos valores realizadas;
- *insertproportion* (de 0 a 1): Define a proporção de operações de inserção realizadas;
- *scanproportion* (de 0 a 1): Define a proporção de operações de leitura em um intervalo de chaves;
- *requestdistribution* (*zipfian*, *uniform*, *latest* ou *multinomial*): Define o algoritmo de distribuição uniforme utilizado para qualquer decisão, seja qual operação executar ou qual tupla realizar a operação em questão;

Por meio dos parâmetros listados acima, o usuário pode personalizar as cargas de trabalho que deseja submeter ao SGBD que está testando. Porém, também é possível utilizar os *workloads* presentes no *YCSB Core Package*. Esse pacote oferece seis *workloads*, os quais são (COOPER *et al.*, 2010):

- *workloadA*: Tem 50% de operações de leitura e 50% de operações de atualização. É considerada uma carga de trabalho de atualização intensiva e pode ser comparada a um aplicativo que atualiza o estado da sessão do usuário;
- *workloadB*: Tem 95% de operações de leitura e 5% de atualizações. É considerada uma carga de trabalho de leitura majoritária e pode ser comparada a um processo de adicionar/ler marcações de pessoas presentes em uma foto. Adicionar uma marcação é uma operação de atualização, mas na maior parte das vezes é feita somente a leitura das marcações;
- *workloadC*: Tem 100% das operações sendo de leitura. É uma carga de somente leitura e pode ser comparada ao processo de obter as informações do perfil de um usuário;
- *workloadD*: Tem 95% de operações de leitura e 5% de atualizações, porém utiliza *requestdistribution* = *latest*. É uma carga que executa leitura nos dados mais recentes, pode ser comparada a atualização de status do usuário em uma rede social;
- *workloadE*: Tem 95% de operações de *Scan* e 5% de inserções. Pode ser comparada a um aplicativo de conversas aninhadas, no qual cada *scan* corresponde às respostas de uma conversa;

- *workloadF*: Tem 95% de operações de leitura e 5% de leituras seguidas de modificações. Pode ser comparada a leitura e modificação de dados de um usuário no banco de dados;

A execução do *benchmark* YCSB é feita por meio da ferramenta *YCSB Client*, a qual consiste em códigos escritos em Java. Segundo Cooper *et al.* (2010), este código foi pensado para permitir que o usuário configure qual banco de dados deseja submeter aos testes, quais *workloads* executar, definir novos *workloads* e customizar parâmetros dessas cargas de trabalho. O processo de *benchmarking* é dividido em duas fases: fase de carregamento na qual os dados para teste são inseridos no banco de dados e fase transacional na qual os testes são submetidos à esse banco.

# 4 EXPERIMENTOS E RESULTADOS

Este capítulo apresenta os experimentos realizados para validar se o Google Cloud Spannner satisfaz as características definidas como critérios para um SGBD pertencer à categoria NewSQL. Tais caracteristicas foram detalhadas na subseção 2.3.1 e são listadas no Quadro 7. A seção 4.1 detalha todo o ambiente experimental no qual foram submetidos os testes. As seções subsequentes agrupam características comuns do Quadro 7, apresentando os respectivos experimentos e resultados obtidos.

**Quadro 7 – Características NewSQL e como o Spanner será avaliado**

| | **Característica** | **NewSQL** | **Como o Spanner será avaliado quanto à característica** |
|---|---|---|---|
| 1 | Propriedades ACID | Fornece | Documentação |
| 2 | Suporte à consultas complexas (junções, por exemplo) | Sim | Experimentação |
| 3 | Processamento OLTP | Eficiente | Experimentação |
| 4 | Suporte a SQL | Sim (Variável) | Documentação e experimentação |
| 5 | Escalabilidade horizontal | Sim | Documentação e experimentação |
| 6 | Naturalmente distribuído | Sim | Documentação |
| 7 | Suporte a consistência forte | Sim | Documentação e experimentação |
| 8 | Depende de esquema | Sim | Documentação e experimentação |
| 9 | Suporte a alto fluxo de dados | Bom | Experimentação |
| 10 | *Delay* por *lock* | Eventualmente | Experimentação |
| 11 | Modelo de armazenamento | Relacional, multi-modelos | Documentação e experimentação |

**Fonte: Autoria própria (2023)**

## 4.1 Ambiente experimental

Para realização dos experimentos foi utilizada a versão do Google Cloud Spanner lançada dia 06 de setembro de 2023. Foram criadas e utilizadas duas instâncias distintas para aplicação dos testes. A primeira, a qual será referenciada como *instância regional*, foi criada na região *southamerica-east1*, contendo 3 réplicas de leitura e gravação localizadas em 3 zonas distintas no estado de São Paulo. Essa instância foi configurada para ter a capacidade de computação de 5000 unidades de processamento, o que equivale a 5 nós distribuídos nos 3 *datacenters*.

A segunda instância utilizada nos testes, que será chamada de *instância multi-regional*, foi criada abrangendo as regiões *nam-eur-asia3* (infelizmente, não estava disponível uma configuração multi-região com a América do Sul). Esta instância executou com 2 réplicas de leitura/gravação na região *us-central1* no estado de Iowa e 2 réplicas de leitura/gravação na região *us-east1* no estado de Carolina do Sul. Além disso, estavam disponíveis 2 réplicas de somente leitura na região *asia-east1* em Taiwan e 1 réplica de somente leitura em cada uma das regiões:

*us-central1* no estado de Oklahoma; *europe-west1* na Bélgica e *europe-west4* nos Países Baixos. Assim como na *instância regional*, a *instância multi-regional* também foi configurada com a capacidade de computação de 5000 unidades de processamento, divididas entre os 9 nós disponíveis. A criação de ambas as instâncias pode ser observada no Apêndice A.

Quanto ao *benchmark* YCSB empregou-se a versão 0.17.0. Utilizou-se um computador com processador AMD Ryzen 5 2600 e 16GB de memória RAM para fazer as requisições ao SGBD Google Cloud Spanner. As especificações deste computador são relevantes pois afetam a quantidade de *threads* que podem ser executadas em paralelo, assim como o tempo para serem concluídas. É importante destacar que essas especificações não afetam o *throughput* do SGBD em si, pois o mesmo é executado em nuvem. Somente as requisições que são enviadas ao SGBD e a avaliação das respostas são feitas em um computador local. A *instância regional* e a *instância multi-regional* foram carregadas com a mesma quantidade de dados (apresentada mais adiante, na seção atual). A fase de carregamento foi realizada pelo comando ilustrado no Quadro 8, que utiliza o termo load associado ao SGBD Google Cloud Spanner.

**Quadro 8 – Carregamento dos dados do YCSB no Spanner**

| |
|---|
| ./bin/ycsb load cloudspanner -P cloudspanner/conf/cloudspanner.properties -P workloads/workloada -p recordcount=5000000 -p cloudspanner.batchinserts=1000 -threads 390 -s |

**Fonte: Autoria própria (2023)**

Conforme observado na Figura 15, os dados foram carregados para execução do *workloada*, pois todos os *worloads* exceto o *E* geram e utilizam o mesmo tipo de dado. O parâmetro *recordcount* indica quantos registros serão gravados. A quantidade inserida equivale a aproximadamente 5GB, pois as instâncias existiram simultaneamente e o período de teste do Spanner limita a 10GB de dados no total. Ainda considerando o Quadro 8, o parâmetro *batchinserts* define quantas inserções serão feitas a cada solicitação de *commit*, sendo configurado conforme recomendação da documentação do *benchmark* YCSB (2023) com o valor 1000. Já o parâmetro *threads* define quantas *threads* serão executadas em paralelo. O valor foi definido para 390 pois o limite de *threads* simultâneas suportadas pelo computador utilizado é de 400.

Para a fase transacional do benchmark, o comando principal utilizado é o *run* (exemplos ilustrados na Quadro 9), juntamente as definições de qual SGBD e *workload* será executado. Nesta fase, o parâmetro *recordcount* indica quantas tuplas serão afetadas pela execução. Outro parâmetro disponível é o *operationcount* que define quantas operações serão realizadas durante a execução. Essa fase também possui o parâmetro *threads*, o qual define quantas *threads* serão executadas emulando o papel de vários clientes simultâneos.

Os experimentos das seções 4.4, 4.5 e 4.6 que envolvem o *benchmark* YCSB utilizaram os parâmetros *recordcount* = 1000000, *operationcount* = 1000000 e *threads* = 390. Os valores foram definidos de forma arbitrária, com execuções de teste com valores distintos (como

5000000 para *recordcount* e *operationcount* por exemplo) e os resultados apresentados não foram divergentes. Portanto, pela quantidade de execuções que foram necessárias e o risco de ultrapassar os créditos do teste, foram escolhidos os valores apresentados para *recordcount* e *operationcount*. Já o valor definido para *threads* respeita o máximo do computador utilizado para fazer as requisições, que é de 400. Foram aplicados os mesmos parâmetros nas 3 cargas de trabalho para que fosse possível a comparação entre as mesmas. Além dos *workloads* presentes no *YCSB Core Package* foi empregada uma carga de trabalho com parâmetros personalizados, referenciada como *workload G*, com os seguintes parâmetros: 95% de atualização, 5% de leitura. A execução das cargas de trabalho transacionais foi realizada conforme os comandos do Quadro 9.

**Quadro 9 – Comandos para execução dos workloads**

**Workload A**

```
./bin/ycsb run cloudspanner -P cloudspanner/conf/cloudspanner.properties -P
workloads/workloada -p recordcount=1000000 -p operationcount=1000000 -threads 390 -s
-p exportfile=D:\dev\ycsb-result\out-wkla.txt
```

**Workload C**

```
./bin/ycsb run cloudspanner -P cloudspanner/conf/cloudspanner.properties -P
workloads/workloadc -p recordcount=1000000 -p operationcount=1000000 -threads 390 -s
-p exportfile=D:\dev\ycsb-result\out2.txt
```

**Workload G**

```
./bin/ycsb run cloudspanner -P cloudspanner/conf/cloudspanner.properties -P
workloads/workloada -p recordcount=1000000 -p operationcount=1000000 -threads 390 -s
-p readproportion=0.05 -p updateproportion=0.95 -p
exportfile=D:\dev\ycsb-result\out-wklg.txt
```

**Fonte: Autoria própria (2023)**

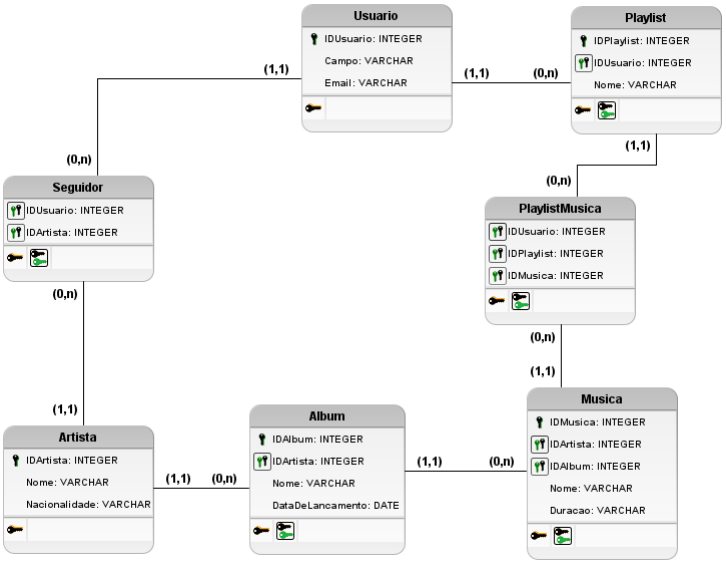
Definido o ambiente experimental, as seções 4.2 até 4.6 apresentarão os experimentos realizados para avaliar se o Google Cloud Spanner atende às características elencadas no Quadro 7 como essenciais para um SGBD pertencer à categoria NewSQL. Com exceção da seção 4.6, as demais condensam características que possuem relação entre si. Dessa forma, os resultados obtidos são capazes de avaliar e analisar mais de uma característica elucidada no Quadro 7 simultaneamente.

## 4.2 Suporte a SQL/Modelo de armazenamento/Dependência de esquema/Consultas complexas

Para avaliar se o Google Cloud Spanner é compatível com as delimitações impostas pelo Quadro 7 quanto ao *suporte à linguagem SQL*, *modelo de armazenamento*, *dependência de esquema* e *suporte a consultas complexas* foram utilizadas tanto a documentação da solução

quanto o trabalho de Corbett *et al.* (2013), o qual apresentou a solução de armazenamento da Google ao mercado. Além disso, as características foram avaliadas de forma prática por meio da criação de um banco de dados na *instância multi-regional*. Ao banco de dados foram submetidas instruções SQL de modo a representar o esquema ilustrado na Figura 14.

**Figura 14 – Esquema utilizado nos experimentos**



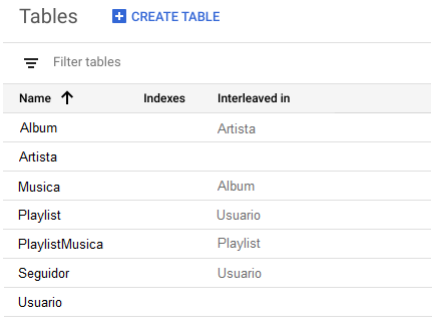

**Fonte: Autoria própria (2023)**

Segundo Corbett *et al.* (2013), o Google Cloud Spanner foi projetado para fazer uso de uma linguagem de consulta que "se parece com o SQL com algumas extensões para suportar campos com valor de *buffer* de protocolo" (CORBETT *et al.*, 2013). Assim, foi construída uma "ramificação" do SQL com algumas sintaxes adicionais para suportar as funcionalidades extras, como leituras em *timestamps* específicos e o entrelaçamento de tabelas para organiza-las fisicamente (com comando INTERLEAVE IN PARENT). No Apêndice C podem ser visualizados alguns comandos SQL do Google Cloud Spanner que diferem dos utilizados no SQL convencional. Para criar o esquema representado na Figura 14 foram utilizados comandos SQL, como o CREATE com INTERLEAVE IN PARENT por exemplo. Portanto, conclui-se que o Spanner oferece *suporte a SQL*.

Sobre o *modelo de armazenamento* é possível afirmar que o Google Cloud Spanner é uma solução multi-modelo. Esta característica foi abordada na subseção 3.2.3 , na qual foi destacado que o Spanner utiliza o modelo chave-valor para armazenamento efetivo dos dados enquanto permite que o usuário os manipule por meio do modelo relacional. Durante a fase de criação das tabelas e carregamento dos dados, foi possível observar que os dados são sempre representados no modelo relacional (a Figura 15 evidencia a representação das tabelas na interface do SGBD) e que não é possível inserir dados utilizando outro modelo, como por exemplo o chave-valor.

Quanto à característica de *dependência de esquema*, Corbett *et al.* (2013) destacam que o Google Cloud Spanner é um SGBD com "tabelas esquematizadas semi-relacionais". O

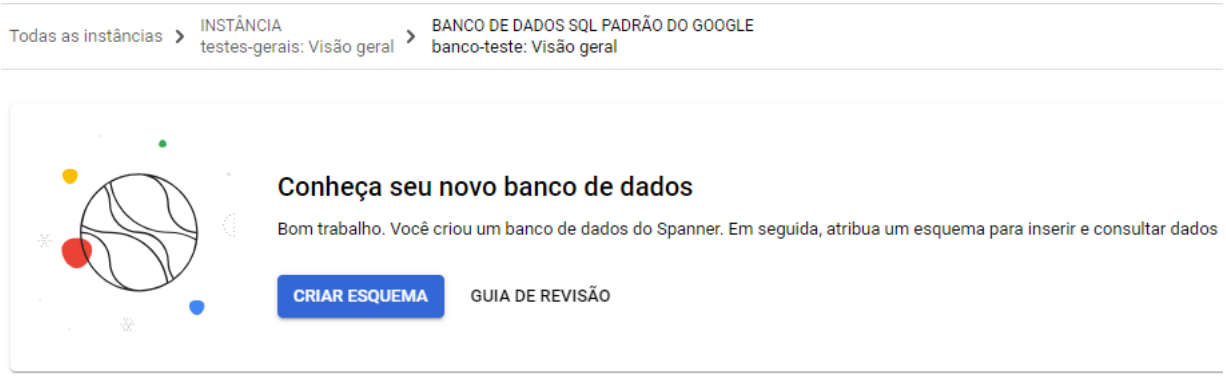
**Figura 15 – Tabelas na interface do Spanner**




**Fonte: Autoria própria (2023)**

termo semi-relacionais faz referência a possibilidade de intercalar as tabelas com comando INTERLEAVE IN PARENT, algo que difere das soluções relacionais comuns. Além disso, na documentação oficial do Spanner é mencionado que os dados da base são fortemente tipados. É necessário definir um esquema para o banco de dados, especificando o tipo de cada dado das colunas das tabelas Spanner (2023a). Na Figura 16 é possível visualizar a mensagem exibida ao criar um novo banco de dados no Spanner. Informando que é necessário criar um esquema antes de começar a utilizar o banco de dados. Assim, é possível afirmar que o Spanner possui *dependência de esquema*.

**Figura 16 – Mensagem solicitando criação de esquema**




**Fonte: Autoria própria (2023)**

Com o objetivo de avaliar se o Google Cloud Spanner oferece *suporte a consultas complexas*, foram submetidas *queries* utilizando o operador de junção. Este tem a característica de tornar mais custoso o processo de resposta por parte do SGBD. Além disso, a execução distribuída em diferentes nós do *cluster* dificulta a construção da resposta da consulta submetida pelo usuário. O experimento realizado consiste de 4 junções distintas, envolvendo um número crescente de tabelas para analisar como o Spanner se comporta a medida que o número de tabelas aumenta na consulta. Cada consulta foi submetida 7 vezes ao banco de dados. Todas elas estão disponíveis na Figura 17.

A primeira consulta realiza uma operação de seleção trivial, recuperando os dados de apenas uma tabela sem aplicar uma junção. A consulta foi bem sucedida nas 7 execuções, in-

**Figura 17 – Junções submetidas ao banco de dados**

Consulta 1:

```
SELECT
  nome,
  email
FROM
  Usuario;
```

Consulta 2:

```
SELECT
  Album.nome AS Album,
  Album.DataDeLancamento AS LancadoEm,
  Artista.nome AS Artista
FROM
  Artista
JOIN
  Album
ON
  Artista.IDArtista = Album.IDArtista;
```

Consulta 3:

```
SELECT
  Musica.nome AS NomeDaMusica,
  NomeDoAlbum,
  NomeDoArtista
FROM (
  SELECT
    Artista.IDArtista AS IDDoArtista,
    Artista.nome AS NomeDoArtista,
    Album.nome AS NomeDoAlbum
  FROM
    Artista
  JOIN
    Album
  ON
    Artista.IDArtista = Album.IDArtista)
JOIN
  Musica
ON
  IDDoArtista = Musica.IDArtista
```

Consulta 4:

```
SELECT
  NomeDaPlaylist,
  NomeDaMusica,
  NomeDoAlbum,
  NomeDoUsuario
FROM (
  SELECT
    Playlist.IDPlaylist AS IdDaPlaylist,
    Usuario.nome AS NomeDoUsuario,
    Playlist.nome AS NomeDaPlaylist
  FROM
    Playlist
  JOIN
    Usuario
  ON
    Usuario.IDUsuario = Playlist.IDUsuario )
LEFT JOIN (
  SELECT
    PlaylistMusica.IDPlaylist AS IdDaPlaylistFromRight,
    NomeDaMusica,
    IdDaMusica,
    NomeDoAlbum
  FROM (
    SELECT
      Musica.nome AS NomeDaMusica,
      Musica.IDMusica AS IdDaMusica,
      Album.nome AS NomeDoAlbum,
    FROM
      Musica
    JOIN
      Album
    ON
      Musica.IDAlbum = Album.IDAlbum )
  JOIN
    PlaylistMusica
  ON
    PlaylistMusica.IDMusica = IdDaMusica )
ON
  IdDaPlaylistFromRight = IdDaPlaylist
ORDER BY
  NomeDaPlaylist;
```

**Fonte: Autoria própria (2023)**

formando os dados dos 10 usuários previamente criados. A resposta obtida pode ser observada na Figura 18.

**Figura 18 – Resultado da Consulta 1**

| nome | email |
|---|---|
| João | joao@email.com |
| Maria | maria@email.com |
| Pedro | pedro@email.com |
| José Carlos | jose2@email.com |
| Ana Paula | ana_paula@gmail.com |
| Joana | joana@email.com |
| Roberto Mariano | robert@gmail.com |
| Carlos Alberto | carlos_alb@email.com |
| Carla Rodriguez | carla_ro@gmail.com |
| Mariana Costa | mariana_costa@gmail.com |

**Fonte: Autoria própria (2023)**

A *Consulta 2* realiza uma consulta aplicando o operador de junção entre as tabelas Artista e Album. Esta foi bem sucedida nas 7 execuções, apresentando 147 resultados associando os álbuns a seus respectivos criadores, assim como a data de lançamento. Como foram inseridos 147 álbuns, o resultado foi o esperado e a resposta com os 10 primeiros resultados pode ser analisada na Figura 19.

A *Consulta 3* realiza uma junção em 3 tabelas, envolvendo as tabelas Artista, Album e Musica. Feita em dois passos, primeiro é realizada a junção entre as tabelas Artista e Album e na sequência o resultado é submetido a uma junção com a tabela Musica. Obteve sucesso nas 7 execuções apresentando 174 resultados, que é o número de músicas inseridas na base de dados, retornando o resultado esperado. Os 10 últimos resultados podem ser analisados na Figura 20.

**Figura 19 – Resultado da Consulta 2**

| Album | LancadoEm | Artista |
|---|---|---|
| Hollywood's Bleeding | 2019-09-06 | Post Malone |
| Beerbongs & Bentleys | 2018-04-27 | Post Malone |
| Stoney | 2016-12-09 | Post Malone |
| Austin | 2023-07-23 | Post Malone |
| Positions | 2020-10-30 | Ariana Grande |
| Thank U, Next | 2019-02-08 | Ariana Grande |
| Sweetener | 2018-08-17 | Ariana Grande |
| Certified Lover Boy | 2021-09-03 | Drake |
| Scorpion | 2018-06-29 | Drake |
| Views | 2016-04-29 | Drake |

**Fonte: Autoria própria (2023)**

**Figura 20 – Resultado da Consulta 3**

| NomeDaMusica | NomeDoAlbum | NomeDoArtista |
|---|---|---|
| Stardust | Stars Dance | Selena Gomez |
| Wonderland | Wonder | Shawn Mendes |
| Lost in You | Wonder | Shawn Mendes |
| Miracle | Wonder | Shawn Mendes |
| Wonderland | Shawn Mendes | Shawn Mendes |
| Lost in You | Shawn Mendes | Shawn Mendes |
| Miracle | Shawn Mendes | Shawn Mendes |
| Wonderland | Illuminate | Shawn Mendes |
| Lost in You | Illuminate | Shawn Mendes |
| Miracle | Illuminate | Shawn Mendes |

**Fonte: Autoria própria (2023)**

Por fim, a *Consulta 4* realiza 4 operações de junção, envolvendo as tabelas Album, Musica, Playlist, PlaylistMusica e Usuario. Além das junções utilizando o *JOIN*, há uma utilizando o *LEFT JOIN*, criando um resultado no qual mesmo *playlists* que não possuem músicas aparecem nos resultados. Todas as execuções foram bem sucedidas e forneceram 85 respostas, sendo este o número de *playlists* previamente inseridas. A Figura 21 apresenta 10 tuplas da resposta, na qual é possível observar o resultado da junção externa à esquerda, com algumas *playlists* aparecendo sem dados preenchidos.

**Figura 21 – Resultado da Consulta 4**

| NomeDaPlaylist | NomeDaMusica | NomeDoAlbum | NomeDoUsuario |
|---|---|---|---|
| Playlist 20 | | | Carlos Alberto |
| Playlist 21 | | | Carlos Alberto |
| Playlist 22 | | | Carla Rodriguez |
| Playlist 23 | Rockstar | Beerbongs & Bentleys | Mariana Costa |
| Playlist 23 | Bad Guy | When We All Fall Asleep, Where Do We Go? | Mariana Costa |
| Playlist 23 | Vete | El Último Tour Del Mundo | Mariana Costa |
| Playlist 23 | Yo Perreo Sola | YHLQMDLG | Mariana Costa |
| Playlist 23 | Dile Que Tú Me Quieres | Aura | Mariana Costa |
| Playlist 23 | Don't Start Now | Dua Lipa | Mariana Costa |
| Playlist 23 | 24K Magic | 24K Magic | Mariana Costa |

**Fonte: Autoria própria (2023)**

## 4.3 Propriedades ACID/Suporte a consistência forte

A documentação do Google Cloud Spanner afirma que tal SGBD garante por completo as propriedades ACID (SPANNER, 2023b). A propriedade de *Atomicidade* é fornecida pela aplicação do protocolo *two phase commit*, em conjunto com o algoritmo paxos, no qual os *coordenadores* garantem a completude ou cancelamento das transações. A propriedade de *Consistência* tem como base o protocolo *two phase commit* e a API *TrueTime*, que garante a não sobreposição dos *timestamps* em multiplas transações. A propriedade de *Isolamento* é garantida pelo protocolo *two phase locking*, assegurando que cada tupla seja modificada por apenas uma transação por vez. Por fim, a propriedade de *Durabilidade* é alcançada por meio da replicação dos dados em um número configurável de *clusters*, criando redundância.

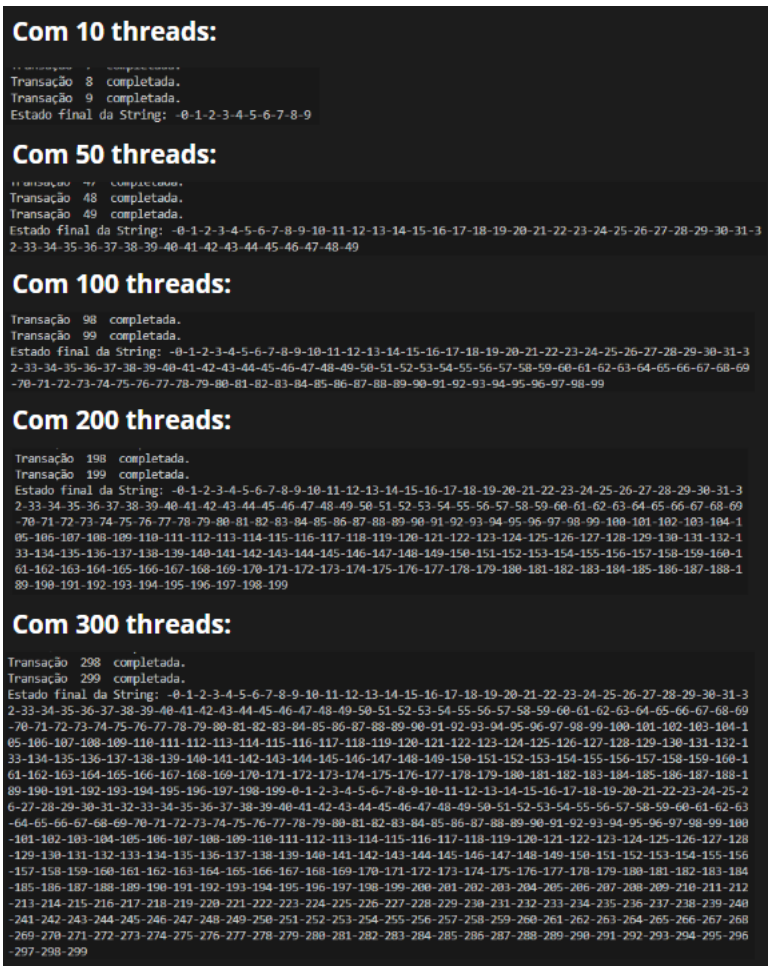
Para avaliar o comportamento do Google Cloud Spanner em relação à característica de *suporte a consistência forte*, foi realizado um experimento utilizando a *instância multi-regional* e um *script* em Python. Em resumo, o experimento consistiu na execução paralela de *n threads* de uma função chamada *read_and_update_field*, responsável por realizar a leitura do campo *fieldx* (no qual o valor de *x* varia de 0 a 4, dependendo de quantas *threads* estão sendo executadas) da tabela *Consistency*, com *id* é igual a 1. Após a leitura, a função concatena o número da *thread* ao final do valor lido e o grava novamente. Em resumo, é executada uma transação de leitura-escrita para cada *thread*, com o objetivo de avaliar se os valores gravados correspondem à sequência de acionamento das *threads*. Ao final, é exibido o estado final do campo. Foram realizadas 5 execuções do código com 10, 50, 100, 200 e 300 *threads* respectivamente, e o valor de *x* que determina qual campo será afetado varia em cada execução, começando em 0. O código relativo a este experimento é apresentado no Apêndice C.

Todas as 5 execuções foram bem-sucedidas, pois todas realizaram a inserção ordenada dos números das *threads* em seus respectivos campos. A Figura 22 compila os resultados dos experimentos realizados. Com as informações obtidas da documentação e os resultados gerados pelo experimento, é possível concluir que o Google Cloud Spanner é capaz de fornecer consistência mesmo em um ambiente distribuído. Isso se evidencia pelo fato de que, mesmo com *threads* alterando simultaneamente o mesmo dado, este refletiu a sequência esperada de atualizações.

## 4.4 Naturalmente distribuído/Escalabilidade horizontal

Segundo Corbett *et al.* (2013), o Google Cloud Spanner foi projetado como um banco de dados "globalmente distribuído". Essa característica fica evidente durante o processo de criação de instâncias, no qual mesmo a *instância regional* é distribuída em *datacenters* localizados na mesma região. Essa informação pode ser visualizada no canto superior direito da criação de cada instância, conforme detalhado no Apêndice A. Além disso, o sistema permite não apenas a definição da quantidade de nós a serem utilizados na instância, mas também possibilita a

**Figura 22 – Resultados do experimento de consistência**



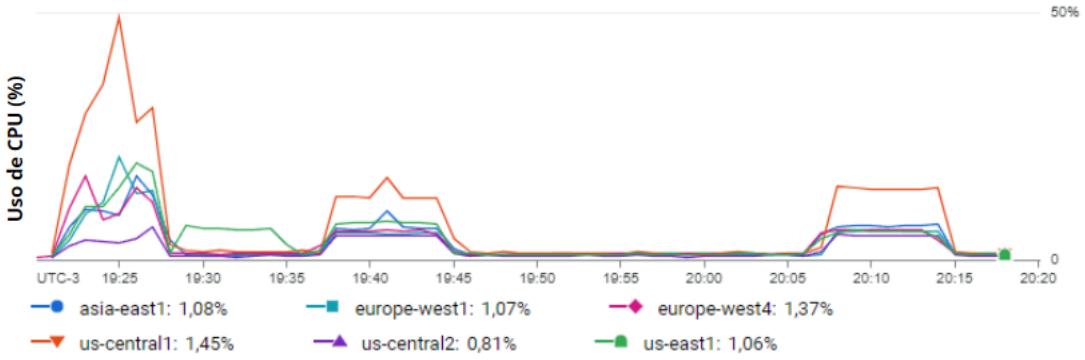

**Fonte: Autoria própria (2023)**

alteração desse número de forma dinâmica, mesmo durante o funcionamento do SGBD. Essa capacidade de adaptação é facilitada pela tarefa *movedir*, como descrito na subseção 3.2.3.

Para avaliar as características da execução distribuída e suporte à *escalabilidade horizontal*, foram realizados dois experimentos. O primeiro envolveu a observação da distribuição das tarefas entre os nós da *instância multi-regional* durante a fase de carregamento do *benchmark* YCSB e seus *workloads a*, *c* e *g*. O gráfico que exibe a distribuição das tarefas foi gerado usando a ferramenta de análise fornecida pelo próprio Google Cloud Spanner. A Figura 23 apresenta os resultados desse experimento, destacando que todas as *zones* estiveram envolvidas em momentos distintos do experimento, confirmando assim, a capacidade do Google Cloud Spanner de operar de maneira distribuída.

O segundo experimento objetivou avaliar a capacidade do Google Cloud Spanner fornecer *escalabilidade horizontal* à medida que o número de nós aumenta. Para realizar esse experimento, o *workload g* foi executado na *instância multi-regional*, inicialmente configura com 1000 unidades de processamento (equivalente a 1 nó). Na metade do experimento, essa configuração foi aumentada para 5000 unidades de processamento com o intuito de verificar se

**Figura 23 – Distribuição das tarefas entre os nós da instância**

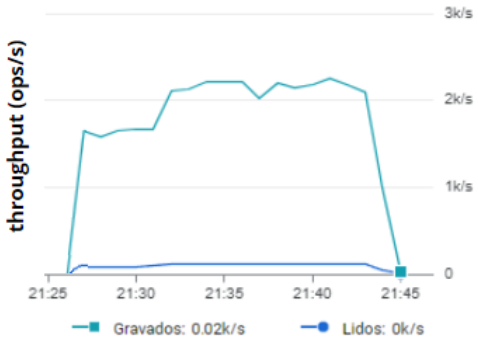



**Fonte: Autoria própria (2023)**

isto teria efeito no *throughput* do SGBD. Novamente, a ferramenta de análise do Google Cloud Spanner foi utilizada, pois fornece informações sobre o *throughput* ao longo do tempo.

A Figura 24 apresenta os resultados obtidos em duas linhas, ilustrando o *throughput* das operações de leitura e das operações de atualização, nomeadas de "Lidos" e "Gravados", respectivamente. Como a carga de trabalho utilizada é majoritariamente de atualização, esta operação apresentou valores superiores (afinal, 95% das solicitações recebidas a cada segundo foram deste tipo). O experimento iniciou pouco depois das 21:25 com 1 nó disponível. Entre 21:30 e 21:35, foi realizado o aumento para 5 nós, resultando em um visível incremento do *throughput*. Essa alteração afetou principalmente as operações de atualização, passando das 2000 operações por segundo, mantendo este valor até o final da carga de trabalho.

**Figura 24 – throughput de 1 a 5 nós**




**Fonte: Autoria própria (2023)**

Com os resultados obtidos e representados na Figura 24, é possível afirmar que o aumento de nós teve um impacto positivo no *throughput* da solução, evidenciando a característica de *escalabilidade horizontal*. Além disso, o primeiro experimento (com resultado ilustrado na Figura 23) demonstrou que as tarefas são divididas entre os nós que compõem o *cluster*, provando que o Google Cloud Spanner também é *naturalmente distribuído*.

## 4.5 Processamento OLTP/Alto fluxo de dados

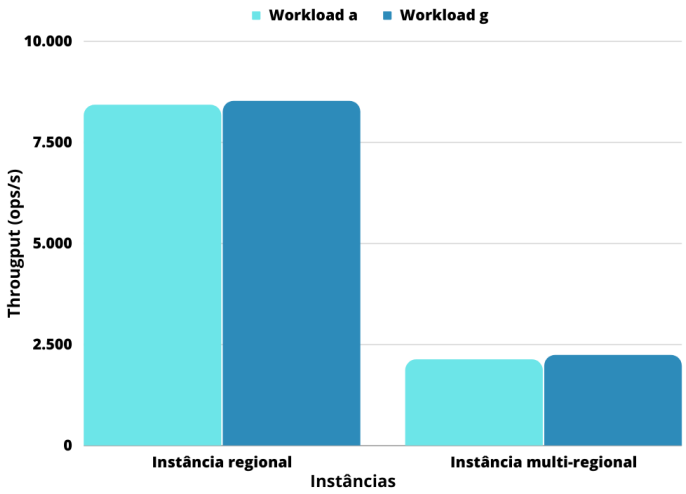
O Google Cloud Spanner possui características fundamentais, como sua natureza distribuída, que o tornam uma solução promissora com suporte a *alto fluxo de dados* e *processamento de transações OLTP*. Além disso, a capacidade de implementar replicação global e facilitar o acesso em todas as regiões do mundo é um fator adicional que contribui para aumentar o fluxo de dados.

Para avaliar essas características, a comparação com um SGBD relacional clássico que não opera de forma distribuída teria sido a abordagem ideal. No entanto, devido ao foco exclusivo no estudo do Google Cloud Spanner, essa comparação não foi realizada. Portanto, para avaliar o comportamento do Google Cloud Spanner em relação a um *alto fluxo de dados* e ao processamento de cargas de trabalho OLTP, foi realizado um experimento utilizando cargas de trabalho do *benchmark* YCSB, pois este foi inicialmente proposto para avaliar soluções NoSQL caracterizadas pelo *processamento OLTP*. Tal experimento utilizou o *workload a* e *workload g* com os parâmetros definidos na **??**, tanto para a *instância regional* quanto para a *instância multi-regional*. A escolha desses dois *workloads* se deu pela característica principal das transações OLTP tratarem operações de escritas e atualizações de curta duração.

Como resultados gerados foram coletados o *throughput* médio e a latência média das operações de cada *workload* do *benchmark* YCSB, representados na Figura 25 e Figura 26 respectivamente. Tais resultados revelam semelhança entre os *throughputs* dos *workloads a* e *g* se comparados os valores obtidos dentro da mesma instância, o que evidencia a capacidade do Google Cloud Spanner em atender uma alta taxa de atualizações. No entanto, se compararmos os *throughputs* obtidos pelos *workloads* na *instância regional* frente aos obtidos na *instância multi-regional*, observa-se uma discrepância nos valores.

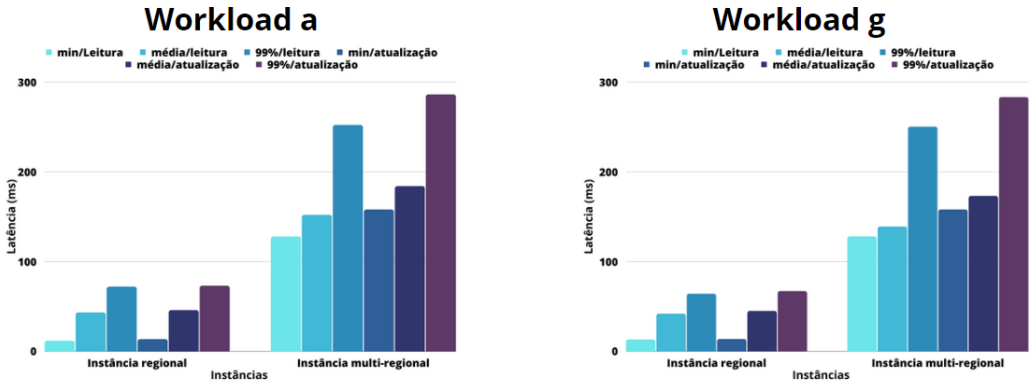
A Figura 26 oferece uma explicação plausível para tal a constatação da discrepância nos valores: as latências nas operações realizadas na *instância multi-regional* são significativamente mais altas do que as observadas na *instância regional*, principalmente se forem observados os valores obtidos em 99% das leituras e atualizações. A latência que existe na comunicação está diretamente relacionada à distância geográfica existente entre o computador que executa as solicitações dos *workloads* e o SGBD. A instância regional foi implantada na região de São Paulo, que fica mais próxima do computador de origem, em comparação com a região de Oklahoma, onde estava localizado o nó central da *instância multi-regional*. Porém, mesmo com as diferenças observadas nos *throughputs*, ambas as instâncias foram capazes de atender às cargas de trabalho submetidas. Em suma, é possível concluir que o Google Cloud Spanner é capaz de satisfazer o processamento OLTP e que a disponibilidade global do SGBD, com o objetivo de reduzir a latência para todos os usuários, é um aspecto a ser considerado no desenvolvimento de uma aplicação.

**Figura 25 – Throughput médio nos workloads a e g**




**Fonte: Autoria própria (2023)**

**Figura 26 – Latências dos workloads a e g**




**Fonte: Autoria própria (2023)**

## 4.6 Delay por lock

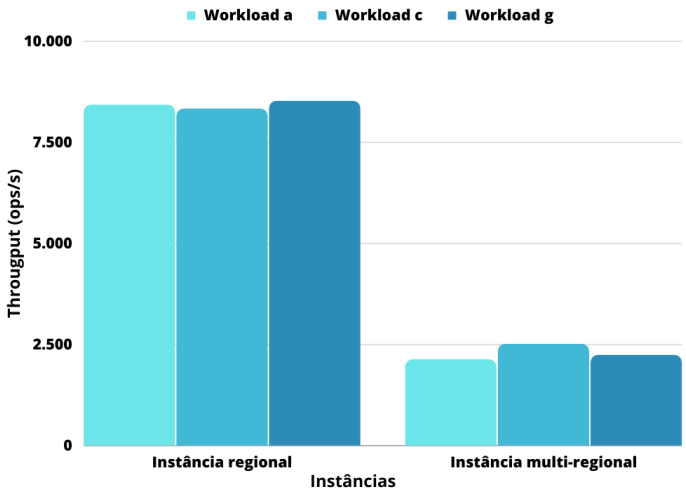
Segundo Corbett *et al.* (2013), transações de somente leitura submetidas ao Google Cloud Spanner são consideradas *lock-free*. Isso implica que essas transações podem ser executadas sem adquirir bloqueio das tuplas envolvidas. No entanto, transações que envolvem escrita e atualização dos dados precisam passar pelo algoritmo *two phase locking*, o que pode potencialmente resultar em atrasos na execução dessas transações. Porém, quando uma transação manipula dados que estão em um mesmo grupo paxos, ela pode executar sem passar pelo algoritmo *two phase locking*, pois as propriedades de consistência já são garantidas por esse mecanismo.

Para analisar o comportamento do Google Cloud Spanner quanto ao *delay* na resposta ocasionado pelo *lock* de tuplas, foram realizados dois experimentos. O primeiro envolveu a aplicação dos *workloads a*, *c* e *g* do *benchmark* YCSB com os parâmetros definidos na **??**. Por

meio destes *workloads*, é possível observar a diferença entre cargas de trabalho de somente leitura e cargas com atualização de dados na situação de acessos simultâneos.

Os resultados obtidos para o *throughput* médio das cargas de trabalho estão representados na Figura 27 e as latências experienciadas estão disponíveis na Figura 28. Observa-se que não houve disparidade entre os *throughputs* de uma mesma instância, exceto pelo *workload c* que teve um desempenho ligeiramente superior na *instância multi-regional*. O desempenho semelhante entre os *workloads* pode ser explicado pela forma como foram configurados e aplicados. Dado que eles foram programados para manipular uma quantidade menor de dados em relação ao tamanho total do banco de dados, é possível que apenas algumas tuplas tenham experimentado acesso simultâneo, o que resultaria na ausência de bloqueio *nas transações com atualização de dados*. As latências observadas na Figura 28 possuem um comportamento semelhante às observadas no experimento da seção 4.5 e a explicação para a disparidade observada nas latências é a mesma: distância geográfica entre as duas instâncias.

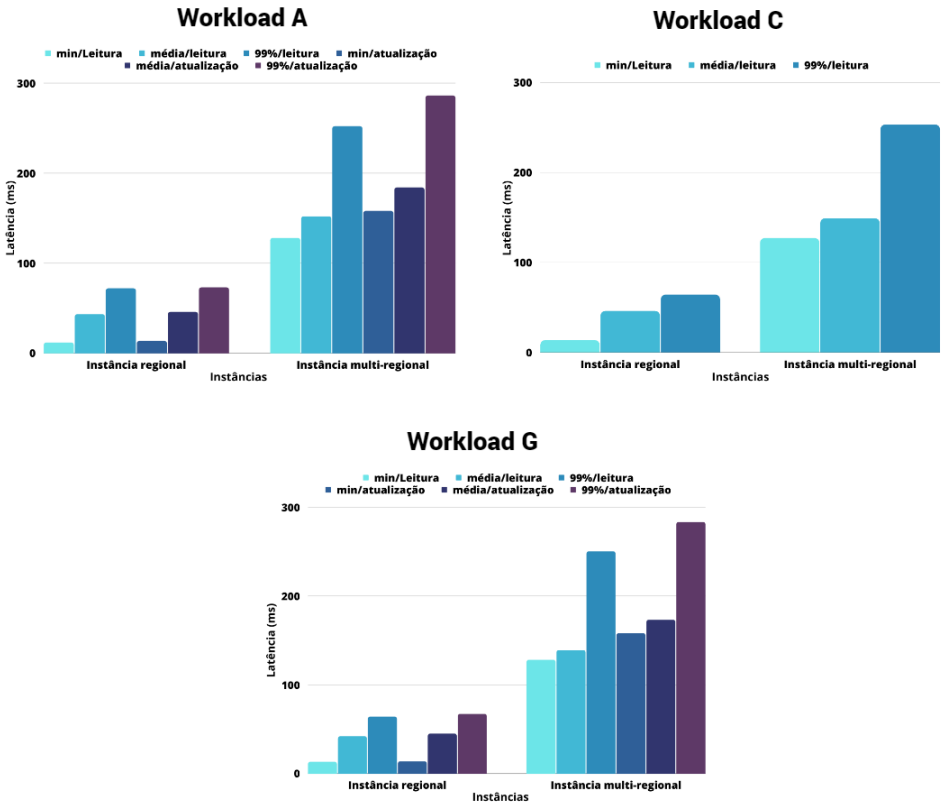
**Figura 27 – Throughput médio nos workloads a, c e g**




**Fonte: Autoria própria (2023)**

No segundo experimento, foi executado o segundo algoritmo Python disponível no Apêndice C, referenciando a *instância multi-regional*. Esse algoritmo utilizou *threads* para simular acesso simultâneo à mesma tupla do banco de dados. Um número *x* de *threads* foram executadas duas vezes: na primeira execução, realizando a atualização do valor do campo *field0* na tabela *Locking* com *id* igual a 1; na segunda execução, realizando apenas a leitura do valor deste campo. Este processo foi implementado pelas funções *read_and_update_field* e *read_only_transaction* definidas em Python, respectivamente. Em ambas as execuções, o tempo para a conclusão das funções foi coletado, e no final, uma média do tempo utilizado em cada uma foi calculada, dividindo o valor total pelo número de *threads*. Foram realizadas execuções do algoritmo, com 10, 50, 100, 200, 300 e 400 *threads*, com três execuções para cada quantidade de *threads*. Os tempos médios obtidos podem ser observado na Figura 29.

**Figura 28 – Latência obtida nos workloads a, c e g**



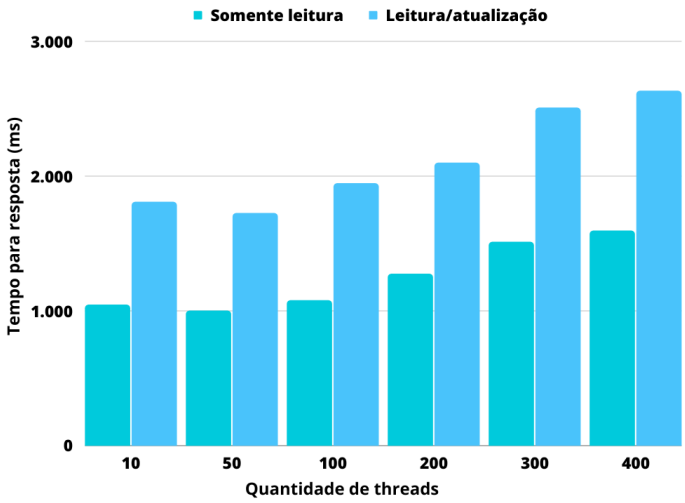

**Fonte: Autoria própria (2023)**

Os resultados disponíveis na Figura 29 indicam uma diferença nos tempos de resposta entre transações de somente leitura e transações de leitura/atualização. Em todos os casos, as transações de somente leitura foram concluídas em menor tempo. Ambos os tipos de transações mostraram aumento no tempo de execução à medida que a quantidade de *threads* aumentava. É importante destacar que, em todas as situações, os tempos de resposta foram relativamente elevados se comparados às latências observadas na Figura 28 referentes ao experimento com o YCSB. No entanto, essa discrepância nas latências pode ser atribuída a diferenças na coleta de tempo entre o código Python desenvolvido e o utilizado pelo YCSB, que é desenvolvido em Java. Apesar dessas particularidades, o comportamento observado está de acordo com o esperado: transações de somente leitura sofrem menos impacto devido ao acesso concorrente aos dados.

Ainda considerando a Figura 29, a discrepância nas latências observadas nas diferentes quantidades de *threads* estão relacionadas às trocas de contexto entre *threads* realizadas pelo processador do computador que realiza as requisições. Assim, pode-se afirmar que o Google Cloud Spanner é afetado pelo atraso causado pelo *lock* de tuplas em situações específicas, como quando há elevação do número de acessos concorrentes à uma mesma tupla, por exemplo.

Por fim, os experimentos realizados e a observação da documentação do Google Cloud Spanner evidenciaram as características elencadas no Quadro 7. A solução analisada con-

**Figura 29 – Tempo de resposta médio do teste de delay por lock**




**Fonte: Autoria própria (2023)**

seguiu combinar características das soluções NoSQL e relacionais. Portanto o Google Cloud Spaner é realmente classificado como uma solução NewSQL.

# 5 CONCLUSÃO

O avanço tecnológico e a crescente demanda por soluções de armazenamento de dados que sejam eficientes e escaláveis têm levado à evolução de diversas abordagens. Inserido neste cenário, este trabalho explorou às soluções de armazenamento de dados presentes no mercado, abordando brevemente as classes relacional e NoSQL e aprofundando-se na categoria NewSQL, que se propõe a combinar as melhores características das classes de armazenamento predecessoras.

Apesar de haver uma gama de trabalhos publicados que abordam o tema NewSQL, foi evidenciada uma carência quanto a definição sólida e uma síntese do que é necessário para uma solução de armazenamento pertencer à categoria NewSQL. Neste contexto, este trabalho, pautado em referencial teórico, agrupou características buscando consolidar e proporcionar maior clareza sobre a categoria de armazenamento NewSQL. Desta forma, solidificando e oferecendo clareza para ela. Além disso, tais características foram evidenciadas na solução Google Cloud Spanner, por intermédio de um ambiente experimental, ferramenta de *benchmark* e códigos desenvolvidos exclusivamente para avaliação.

Na realização dos experimentos, foram encontradas dificuldades. A política de teste com consumo de crédito do Google Cloud Spanner se mostrou limitada para os experimentos propostos. Instâncias personalizadas (diferentes da instância de teste, que tem somente um nó e 10GB de armazenamento) consumiram os créditos rapidamente. Desta forma, não foi possível empregar parâmetros nos *workloads* do *benchmark* YCSB que de fato colocassem o SGBD em situação crítica quanto ao uso de processamento e gestão do *lock* das tuplas, limitando assim os experimentos realizados. Além disso, foi observado que somente um computador realizando as requisições do YCSB não foi suficiente para "estressar" o SGBD. Todavia, foi possível evidenciar as características definidas como essenciais para classificar o SGBD Google Cloud Spanner como pertencente a categoria NewSQL.

Observou-se que a categoria NewSQL é promissora como uma solução para as necessidades atuais de alta disponibilidade e consistência dos dados. Os resultados dos experimentos evidenciaram que esta categoria de armazenamento depende de escolhas de arquitetura (como quais protocolos utiliza para comunicação entre os nós e quais técnicas aplica para garantir as propriedades ACID). Também depende da modelagem do esquema e principalmente do SGBD estar localizado em regiões próximas aos usuários para fornecer as melhores características dos SGBDs Relacionais e NoSQL. Além disso, também ficou claro que o NewSQL em si não oferece uma revolução tecnológica em relação às classes relacional e NoSQL. Porém combina uma série de tecnologias já existentes e até mesmo aplicadas nas soluções presentes no mercado, como o algoritmo paxos e o protocolo MVCC, como é o exemplo do Google Cloud Spanner.

Por fim, uma sugestão de trabalhos futuros é utilizar as características elencadas neste trabalho para realizar um comparativo entre soluções NewSQL, de modo a avaliar o impacto

gerado pela aplicação de outros algoritmos e protocolos na arquitetura do SGBD. Podendo até mesmo comparar com o Google Cloud Spanner que utiliza o algoritmo paxos e a API TrueTime, por exemplo. Além disso, uma comparação mais aprofundada entre uma solução da categoria NewSQL e uma solução relacional quanto aos aspectos arquiteturais e de desempenho seria importante para avaliar a real diferença entre as duas tecnologias de armazenamento de dados.

# REFERÊNCIAS


ARIF, H. *et al.* A comparison between google cloud service and icloud. *In*: IEEE. **2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS)**. [*S.l.*], 2019. p. 337–340.

ASLETT, M. **How will the database incumbents respond to NoSQL and NewSQL**. 2011. The 451 Group.

ASLETT, M. **What we talk about when we talk about newsql**. 2011. Disponível em: https://blogs.451research.com/information_management/2011/04/06/what-we-talk-about-when-we-talk-about-newsql/. Acesso em: 22 nov. 2022.

ASLETT, M. **Ten years of NewSQL: Back to the future of distributed relational databases**. 2021. Disponível em: https://www.spglobal.com/marketintelligence/en/news-insights/blog/ten-years-of-newsql-back-to-the-future-of-distributed-relational-databases. Acesso em: 22 nov. 2022.

BARTHELS, C. *et al.* Strong consistency is not hard to get: Two-phase locking and two-phase commit on thousands of cores. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 12, n. 13, p. 2325–2338, 2019.

BERG, K.; SEYMOUR, T.; GOEL, R. History of databases. **International Journal of Management and Information Systems**, v. 17, n. 1, p. 29–36, 2012. Disponível em: https://doi.org/10.19030/ijmis.v17i1.7587. Acesso em: 22 nov. 2022.

BERNSTEIN, P. A.; GOODMAN, N. Multiversion concurrency control—theory and algorithms. **ACM Transactions on Database Systems (TODS)**, ACM New York, NY, USA, v. 8, n. 4, p. 465–483, 1983.

BREWER, E. Spanner, truetime and the cap theorem. 2017.

BREWER, E. A. **Towards robust distributed systems**. 2000. PODC.

CHAMBERLIN, D. D. Early history of sql. **IEEE Annals of the History of Computing**, v. 34, n. 4, p. 78–82, 2012.

CHAUDHRY, N.; YOUSAF, M. M. Architectural assessment of nosql and newsql systems. **Distributed and Parallel Databases**, v. 38, n. 4, p. 881–926, 2020.

CHEBOTKO, A.; KASHLEV, A.; LU, S. A big data modeling methodology for apache cassandra. *In*: IEEE. **2015 IEEE International Congress on Big Data**. New York, 2015. p. 238–245.

CHEN, P. The entity-relationship model—toward a unified view of data. **ACM Transactions on Database Systems**, v. 1, n. 1, p. 9–36, 1976.

CHEREJA, I. *et al.* Multidimensional analysis of newsql database systems. *In*: **Lecture Notes in Networks and Systems**. New York: Springer, 2021. p. 221–236.

COCKROACHDB. 2022. Disponível em: https://www.cockroachlabs.com. Acesso em: 04 dez. 2022.

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

COOPER, B. F. *et al.* Benchmarking cloud serving systems with ycsb. *In*: **Proceedings of the 1st ACM symposium on Cloud computing**. [*S.l.*: *s.n.*], 2010. p. 143–154.

CORBETT, J. C. *et al.* Spanner: Google's globally distributed database. **ACM Transactions on Computer Systems (TOCS)**, ACM New York, NY, USA, v. 31, n. 3, p. 1–22, 2013.

DB-ENGINES. **DB-Engines Ranking of Relational DBMS**. 2022. Disponível em: https://db-engines.com/en/ranking/relational+dbms. Acesso em: 04 dez. 2022.

DYNAMODB. **O que é um banco de dados de chave-valor?** 2022. Disponível em: https://aws.amazon.com/pt/nosql/key-value/. Acesso em: 04 dez. 2022.

FEATHERSTON, D. Cassandra: Principles and application. **Department of computer science university of illinois at Urbana-champaign**, 2010.

GROLINGER, K. *et al.* Data management in cloud environments: Nosql and newsql data stores. **Journal of Cloud Computing: Advances, Systems and Applications**, v. 2, n. 1, p. 1–24, 2013. Disponível em: https://doi.org/10.1186/2192-113X-2-22. Acesso em: 22 nov. 2022.

HARDING, R. *et al.* An evaluation of distributed concurrency control. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 10, n. 5, p. 553–564, 2017.

HARRISON, G. **Next Generation Databases**: Nosql, newsql and big data. New York: Apress, 2015.

ISMAIL, W. *et al.* A survey of newsql dbmss focusing on taxonomy, comparison and open issues. **Journal of Computer Science and Computational Mathematics**, v. 11, n. 4, p. 87–95, 2021.

KHASAWNEH, T. N.; AL-SAHLEE, M. H.; SAFIA, A. A. Sql, newsql, and nosql databases: A comparative survey. *In*: **2020 11th International Conference on Information and Communication Systems**. Irbid: ANAIS IEEE, 2020. p. 13–21.

KNOB, R. *et al.* Uma análise de soluções newsql. *In*: **Anais da XV Escola Regional de Banco de Dados**. Porto Alegre, RS, Brasil: SBC, 2019. p. 21–30. ISSN 2595-413X. Disponível em: https://sol.sbc.org.br/index.php/erbd/article/view/8475.

KUMAR, R.; CHARU, S. Newsql databases: Scalable rdbms for oltp needs to handle big data. **International Journal of Modern Computer Science**, v. 3, n. 1, p. 13–17, 2014.

LAMPORT, L. Paxos made simple. **ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)**, p. 51–58, 2001.

LAMPORT, L. The part-time parliament. *In*: **Concurrency: the Works of Leslie Lamport**. [*S.l.*: *s.n.*], 2019. p. 277–317.

LESLIE, L. The part-time parliament. **ACM Trans. on Computer Systems**, v. 16, p. 133–169, 1998.

LIFHJELM, T. **A scalability evaluation on CockroachDB**. 2021. Monografia (Bacharelado em Ciência da Computação) — Department of computing science, Umea University, Umea, 2021.

MO, J. S. **Transactions in NewSQL**. 2022. Dissertação (Mestrado em Ciência da Computacão) — NTNU, 2022.

MOHMMED, A. G. M.; OSMAN, S. E. F. Study on sql vs. nosql vs. newsql. **Journal of Multidisciplinary Engineering Science Studies**, v. 3, n. 6, p. 1821–1823, 2017.

PAVLO, A.; ASLETT, M. What's really new with newsql? **ACM SIGMOD Record**, v. 45, n. 2, p. 45–55, 2016.

RASHEED, Y.; QUTQUT, M. H.; ALMASALHA, F. Overview of the current status of nosql database. **International Journal of Computer Science and Network Security**, v. 19, n. 4, p. 47–53, 2019.

SABHARWAL, N.; EDWARD, S. G. Hands on google cloud sql and cloud spanner. **Deployment, Administration and Use Cases with Python, Apress**, Springer, 2019.

SADALAGE, P. J.; FOWLER, M. **NoSQL distilled: a brief guide to the emerging world of polyglot persistence**. Crawfordsville, Indiana: Pearson Education, 2013.

SAP-HANA. **O que é o SAP HANA?** 2022. Disponível em: https://www.sap.com/brazil/products/technology-platform/hana/what-is-sap-hana.html. Acesso em: 04 dez. 2022.

SCHREINER, G. A. *et al.* Newsql through the looking glass. *In*: **The 21st International Conference on Information Integration and Web-based Applications and Services**. New York: Association for Computing Machinery, 2019. p. 361–369.

SIVAKUMARAN, J.; ALI, S. Z. Rdbms current challenges and oppurtunities with nosql to newsql. **Journal of Student Research**, 2017.

SPANNER, D. **Sobre esquemas**. 2023. Disponível em: https://cloud.google.com/spanner/docs/schema-and-data-model?hl=pt-br. Acesso em: 20 set. 2023.

SPANNER, D. **Sobre transações**. 2023. Disponível em: https://cloud.google.com/spanner/docs/transactions?hl=pt-br. Acesso em: 20 set. 2023.

SPANNER, G. C. **Documentação do Cloud Spanner**. 2023. Disponível em: https://cloud.google.com/spanner/docs. Acesso em: 28 ago. 2023.

STONEBRAKER, M. Sql databases v. nosql databases. **Communications of the ACM**, v. 53, n. 4, p. 10–11, 2010.

SUNYAEV, A.; SUNYAEV, A. Cloud computing. **Internet Computing: Principles of Distributed Systems and Emerging Internet-Based Technologies**, Springer, p. 195–236, 2020.

TAILOR, H.; CHOUDHARY, S.; JAIN, V. **Rise of newsql**. [*S.l.*]: Citeseer, 2014.

TPC, T. P. P. C. **Tpc benchmark e**. 2015. Disponível em: https://www.tpc.org/tpc_documents_current_versions/pdf/tpc-e_v1.14.0.pdf.

VOLTDB. 2022. Disponível em: https://www.voltactivedata.com. Acesso em: 04 dez. 2022.

WEI, X. *et al.* Unifying timestamp with transaction ordering for mvcc with decentralized scalar timestamp. *In*: **NSDI**. [*S.l.*: *s.n.*], 2021. p. 357–372.

WU, Y. *et al.* An empirical evaluation of in-memory multi-version concurrency control. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 10, n. 7, p. 781–792, 2017.

YCSB. **Cloud Spanner Driver for YCSB**. 2023. Disponível em: https://github.com/brianfrankcooper/YCSB/blob/master/cloudspanner/README.md. Acesso em: 01 out. 2023.

ZDEPSKI, C. **PDDM: um método de projeto de banco de dados aplicado à persistência poliglota**. 2019. Dissertação (Mestrado em Ciência da Computacão) — Programa de Pós-Graduação em Ciência da Computacão, Universidade Tecnológica Federal do Paraná, Ponta Grossa, 2019.

# APÊNDICE A – Criação das instâncias

# Instância multi-regional

### Nomeie sua instância

Uma instância tem um **nome** e um **código**. O nome é usado apenas para fins de exibição. O código é um identificador permanente e exclusivo.

**Nome da instância ***
teste-multiregiao
O nome precisa ter de 4 a 30 caracteres

**ID da instância ***
teste-multiregiao
Letras minúsculas, números e hífens permitidos

### Selecione uma configuração

Determina a localização dos seus nós e dados. Afeta custos, desempenho e replicação. Uma configuração multirregional selecionará a região líder padrão para suas réplicas líderes. É possível alterar a região líder a qualquer momento com uma instrução DDL. Saiba mais

COMPARE REGION CONFIGURATIONS

- ○ Regional
- ◉ Multirregional

nam-eur-asia3 (Iowa/Carolina do Sul/Oklahoma/Bélgica/Países Baixos/Taiwan)

CONFIGURAR MAIS RÉPLICAS SOMENTE LEITURA

### Alocar capacidade de computação

A capacidade de computação determina a quantidade de capacidade de processamento de dados, consultas por segundo (QPS) e limites de armazenamento na sua instância. Um nó equivale a mil unidades de processamento. Afeta o faturamento.

**Unidade ***
Unidades de processame...

**Quantidade ***
5000
Somente números inteiros. Insira em incrementos de 100 a 1.000, seguidos por incrementos de 1.000.

### Resumo

O custo de armazenamento depende dos GBs armazenados por mês. O custo de computação refere-se à cobrança por hora de nós ou unidades de processamento da instância.

| | |
|---|---|
| **Configuração** | nam-eur-asia3 (Iowa/Carolina do Sul/Oklahoma/Bélgica/Países Baixos/Taiwan) |
| **Réplicas** | 2 réplicas de leitura-gravação em us-central1 (Iowa) - *região líder padrão*<br>2 réplicas de leitura-gravação em us-east1 (Carolina do Sul)<br>1 ver réplica em us-central2 (Oklahoma) - *região privada do GCP*<br>1 réplica somente leitura em europe-west1 (Bélgica)<br>1 réplica somente leitura em europe-west4 (Países Baixos)<br>2 réplicas somente leitura em asia-east1 (Taiwan) |
| **Disponibilidade** | 99,999% de disponibilidade de SLA |
| **Capacidade máxima de armazenamento** | 20 TB |

| Item | Estimated Cost |
|---|---|
| ▸ **Custo de computação** | US$ 35,00 por hora** |
| ▸ **Custo de armazenamento** | US$ 0,90 por GB/mês |

**Com os descontos por compromisso de uso, você economiza 20% ou 40% com compromissos de um e três anos, respectivamente. Learn more

# Instância regional

### Nomeie sua instância

Uma instância tem um **nome** e um **código**. O nome é usado apenas para fins de exibição. O código é um identificador permanente e exclusivo.

**Nome da instância ***
teste-ycsb
O nome precisa ter de 4 a 30 caracteres

**ID da instância ***
teste-ycsb
Letras minúsculas, números e hífens permitidos

### Selecione uma configuração

Determina a localização dos seus nós e dados. Afeta custos, desempenho e replicação. Uma configuração multirregional selecionará a região líder padrão para suas réplicas líderes. É possível alterar a região líder a qualquer momento com uma instrução DDL. Saiba mais

COMPARE REGION CONFIGURATIONS

- ◉ Regional
- ○ Multirregional

southamerica-east1 (São Paulo)

CONFIGURAR MAIS RÉPLICAS SOMENTE LEITURA

### Alocar capacidade de computação

A capacidade de computação determina a quantidade de capacidade de processamento de dados, consultas por segundo (QPS) e limites de armazenamento na sua instância. Um nó equivale a mil unidades de processamento. Afeta o faturamento.

**Unidade ***
Unidades de processame...

**Quantidade ***
5000
Somente números inteiros. Insira em incrementos de 100 a 1.000, seguidos por incrementos de 1.000.

### Resumo

O custo de armazenamento depende dos GBs armazenados por mês. O custo de computação refere-se à cobrança por hora de nós ou unidades de processamento da instância.

| | |
|---|---|
| **Configuração** | southamerica-east1 (São Paulo) |
| **Réplicas** | 3 réplicas de leitura-gravação em 3 zonas separadas na região southamerica-east1 |
| **Disponibilidade** | 99,99% de disponibilidade de SLA |
| **Capacidade máxima de armazenamento** | 20 TB |

| Item | Estimated Cost |
|---|---|
| ▸ **Custo de computação** | US$ 6,75 por hora** |
| ▸ **Custo de armazenamento** | US$ 0,45 por GB/mês |

**Com os descontos por compromisso de uso, você economiza 20% ou 40% com compromissos de um e três anos, respectivamente. Learn more

# APÊNDICE B – Códigos Python utilizados nos experimentos

O código abaixo foi utilizado para avaliar a característica de *suporte a consistência forte*:

```
import threading
import queue
from google.cloud import spanner

project_id = 'perfect-jetty-398922'
instance_id = 'teste-multiregiao'
database_id = 'teste-consistencia'
table_name = "Consistency"
field_name = "field1"

client = spanner.Client(project=project_id)

num_threads = 50

task_queue = queue.Queue()

def read_and_update_field(thread_number):
    def update_field(transaction):
        field_0_values = transaction.read(
            table=table_name,
            keyset=spanner.KeySet(all_=True),
            columns=(field_name,),
            limit=1,
        )
        field_0_value = list(field_0_values)[0][0]

        new_field_0_value = field_0_value + "-" + str(thread_number)

        transaction.update(
            table=table_name,
            columns=("id", field_name),
            values=[(1, new_field_0_value)],
        )
```

```
    task_queue.put((thread_number, update_field))

threads = []
for i in range(num_threads):
    thread = threading.Thread(target=read_and_update_field,
   args=(i,))
    threads.append(thread)
    thread.start()

for thread in threads:
    thread.join()

instance = client.instance(instance_id)
database = instance.database(database_id)

while not task_queue.empty():
    thread_number, task = task_queue.get()
    print("Transação ", thread_number, " completada.")
    database.run_in_transaction(task)

database = client.instance(instance_id).database(database_id)
snapshot = database.snapshot()

query = f'SELECT {field_name} FROM {table_name} where id = 1'
with database.snapshot() as snapshot:
    results = snapshot.execute_sql(sql=query)
    for row in results:
        print(f'Estado final da String: {row[0]}')
    snapshot.close()
```

Já o código abaixo foi utilizado para avaliar a característica de *delay por lock*:

```
import threading
from google.cloud import spanner
```

```
import time

project_id = 'perfect-jetty-398922'
instance_id = 'teste-multiregiao'
database_id = 'teste-consistencia'
table_name = "Locking"
field_name = "field0"

client = spanner.Client(project=project_id)

num_threads = 400

time_holder = [0, 0]

def read_and_update_field(thread_number):
    instance = client.instance(instance_id)
    database = instance.database(database_id)

    inicio = time.perf_counter()

    def update_field(transaction):
        transaction.read(
            table=table_name,
            keyset=spanner.KeySet(all_=True),
            columns=(field_name,),
            limit=1,
        )
        transaction.update(
            table=table_name,
            columns=("id", field_name),
            values=[(1, 'teste_delay')],
        )

    database.run_in_transaction(update_field)
```

```
    fim = time.perf_counter()
    tempo_decorrido_em_microssegundos = (fim - inicio) * 1e6
    time_holder[0] = tempo_decorrido_em_microssegundos +
   time_holder[0]

    print("Transação ", thread_number, " completada.")

def read_only_transaction(thread_number):
    instance = client.instance(instance_id)
    database = instance.database(database_id)

    inicio = time.perf_counter()

    with database.snapshot() as snapshot:
        results = snapshot.execute_sql(sql = "SELECT * FROM
   ${table_name}")


    fim = time.perf_counter()
    tempo_decorrido_em_microssegundos = (fim - inicio) * 1e6
    time_holder[1] = tempo_decorrido_em_microssegundos +
   time_holder[1]

    print("Transação ", thread_number, " completada.")

threads = []
for i in range(num_threads):
    thread = threading.Thread(target=read_and_update_field,
   args=(i,))
    threads.append(thread)
    thread.start()

for thread in threads:
```

```
    thread.join()

threads2 = []
for i in range(num_threads):
    thread = threading.Thread(target=read_only_transaction,
   args=(i,))
    threads2.append(thread)
    thread.start()

for thread in threads2:
    thread.join()

print("Tempo medio read only: ", time_holder[1] / num_threads,
   "microssegundos\n")
print("Tempo medio read and update: ", time_holder[0] / num_threads,
   "microssegundos\n")
```

# APÊNDICE C – Algumas diferenças entre SQL clássico e o SQL do Google Cloud Spanner

```
-- Criação da tabela usuário no SQL convencional;
CREATE TABLE Usuarios (
    ID INT PRIMARY KEY,
    Nome VARCHAR(100),
    Email VARCHAR(100)
);

-- Criação no SQL do Google Cloud Spanner, onde a definição da chave
   primária fica fora da TABLE;
CREATE TABLE Usuarios (
    ID INT64,
    Nome STRING(100),
    Email STRING(100),
) PRIMARY KEY (ID);

-------------------------------------------------------------------------

-- O Google Cloud Spanner tem os seguintes tipos de dados, que
   diferem dos tipos presentes no SQL convencional:
STRING(x) --Análogo ao VARCHAR
INT64  --Análogo ao INT
FLOAT64 --Análogo ao FLOAT
BYTES(x) --Armazena uma quantidade x de bytes, relativo ao "campos
   com valor de buffer de protocolo" citado por CORBETT et al., 2013.

-------------------------------------------------------------------------
-- Relação Album no SQL convencional
CREATE TABLE Album (
    IDAlbum INT PRIMARY KEY,
    IDArtista INT,
    Nome VARCHAR(50) NOT NULL,
    DataDeLancamento DATE,
    FOREIGN KEY (IDArtista) REFERENCES Artista (IDArtista)
);
```

```
-- Exemplo da mesma relação no Spanner, com o INTERLEAVE IN PARENT
   para colocalização dos dados
CREATE TABLE Album (
    IDAlbum INT64,
    IDArtista INT64,
    Nome STRING(50) NOT NULL,
    DataDeLancamento DATE,
    CONSTRAINT FK_ArtistaAlbum FOREIGN KEY (IDArtista) REFERENCES
   Artista (IDArtista)
) PRIMARY KEY (IDArtista, IDAlbum),
INTERLEAVE IN PARENT Artista ON DELETE CASCADE;

-----------------------------------------------------------------------
-- Por meio do termo OPTIONS (allow_commit_timestamp=true) o Spanner
   permite que o campo em questão seja preenchido automaticamente
   quando a transação for efetivada;
CREATE TABLE Logs (
    ID INT64,
    Mensagem STRING(1000),
    DataHora TIMESTAMP OPTIONS (allow_commit_timestamp=true)
) PRIMARY KEY (ID);

-----------------------------------------------------------------------
-- Por meio do comando abaixo, é possível observar o estado de uma
   tabela no "passado", neste exemplo é retornada a tabela Usuarios
   no datetime = "2022-01-01 00:00:00 UTC";
SELECT * FROM Usuarios
FOR SYSTEM_TIME AS OF TIMESTAMP "2022-01-01 00:00:00 UTC";

-----------------------------------------------------------------------
-- O Spanner permite criar tabelas que armazenam Arrays, como no
   exemplo abaixo:
CREATE TABLE Biblioteca (
```

```
    ID INT64,
    Titulos ARRAY<STRING(100)>
) PRIMARY KEY (ID);

-------------------------------------------------------------------------
 -- O Spanner permite verificar como uma consulta seria
   particionada, como no exemplo abaixo;
PARTITION QUERY SELECT * FROM MinhaTabela WHERE ID > 100;
-------------------------------------------------------------------------
```