

# O NOVO PARADIGMA DE BANCO DE DADOS: Uma análise prática sobre o banco de dados NewSQL

## *THE NEW PARADIGM OF DATABASE: a practical analysis of the NewSQL database*

Raziel Miranda Rodrigues  
Graduando em Banco de Dados pela Fatec Bauru  
E-mail: raziel.rodrigues@fatec.sp.gov.br

Patricia Bellin Ribeiro  
Professor em Banco de Dados pela Fatec Bauru  
E-mail: patricia.ribeiro5@fatec.sp.gov.br

### RESUMO

O *New Structured Query Language (NewSQL)* é um paradigma diferente dos bancos de dados atuais seja ele *Not Only Structured Query Language (NoSQL)* ou *Structured Query Language (SQL)*. Esse tipo de banco de dados está preparado com o melhor dos dois mundos sendo capaz de juntar as propriedades Atomicidade, Consistência, Isolamento e Durabilidade (ACID) do banco de dados *SQL (Structured Query Language)* com as propriedades de escalabilidade e velocidade do banco de dados *NoSQL (Not Only Structured Query Language)*.

Empresas de transações online como bancos e mercado financeiro estão a utilizar esse tipo de banco de dados para a gestão dos seus respectivos *softwares*, esse paradigma ainda é novo, mas mesmo assim conta com diversas documentações de como utilizá-lo e também já existem Sistemas Gerenciadores de Banco de Dados (*SGBD*) prontos para usar esse tipo de paradigma.

Esse artigo tem o objetivo de mostrar na prática como utilizar esse paradigma, desde a instalação de um Sistemas Gerenciadores de Banco de Dados que fornece suporte a esse banco de dados até a utilização das suas funcionalidades e um teste de *benchmarking* que foi realizado em uma máquina Linux utilizando o banco de dados VoltDB onde foi inserido 500 mil de registros em uma tabela usando uma *procedure* e uma aplicação em *JAVA*,

como complemento foi efetuado um novo teste de *benchmarking* em um banco de dados MySQL utilizando a linguagem de programação PHP para a parte da aplicação onde foi obtido resultados para realizar uma comparação entre os dois banco de dados.

**Palavras-chave:** *NewSQL*, Banco de dados OLAP, VoltDB, junção *NoSQL* e *SQL*, *benchmarking NewSQL*.

### ABSTRACT

*New Structured Query Language (NewSQL)* is a different paradigm from current databases whether it is *Not Only Structured Query Language (NoSQL)* or *Structured Query Language (SQL)*. This type of database is prepared with the best of both worlds being able to combine the Atomicity, Consistency, Isolation and Durability (ACID) properties of the *SQL (Structured Query Language)* database with the scalability and speed properties of the database. *NoSQL (Not Only Structured Query Language)* data. Online transaction companies such as banks and the financial market are using this type of database for the management of their respective software, this paradigm is still new, but it still has several documentation on how to use it and there are also Management Systems Database (DBMS) ready to use this type of paradigm. This article aims to show in practice how to use this paradigm, from the installation of a Database Management

*Systems that provides support to this database to the use of its features and a benchmarking test that was performed on a machine Linux using the VoltDB database where 500 thousand records were inserted into a table using a procedure and a JAVA application, as a complement, a new benchmarking test was performed on a MySQL database using the PHP programming language for the part application where results were obtained to make a comparison between the two databases.*

**Keywords:** *NewSQL, OLAP database, VoltDB, NoSQL and SQL junction, benchmarking NewSQL.*

## 1 INTRODUÇÃO

Quando abordamos a respeito de grandes volumes de dados, o antigo modelo relacional de banco de dados o *Structured Query Language* (SQL), pode encontrar certas dificuldades em lidar com a taxa de crescimento de dados, flexibilização de formatação de dados e sobrecarga de desempenho. A rigidez do modo que o banco de dados é construído fornece pouca possibilidade de alterações futuras. Vamos a um exemplo desse cenário:

Uma empresa começa com um certo modelo de negócio e faz uma modelagem inicial do banco de dados para atender a essas regras de negócio, com o tempo a empresa decide mudar algumas regras. Entretanto, essas alterações no banco de dados as vezes não podem ser feitas. Senão, pode causar danos irreversíveis na estrutura de dados e fazer perder relações importantes da tabela X para a Y, pois, elas têm uma relação de *Foreign Key* (FK).

Uma saída foi a criação dos bancos de dados *Not Only Structured Query Language* (NoSQL) que por sua vez são mais rápidos e conseguem lidar com essa massa, entretanto não respeitando as regras de Atomicidade, Consistência, Isolamento e Durabilidade (ACID), também não tendo suporte a Processamento de Transações em Tempo Real (OLTP) e o Processamento de Análises em Tempo Real (OLAP) e também não consegue realizar a análise de dados (agregados, transformar, etc.).

Todavia, como o seu banco de dados não é feito de forma rígida é possível alterar o seu modelo de banco de dados com mais flexibilidade, outro ponto positivo é que se pode mudar o formato de dados com mais tranquilidade e a sua sobrecarga é de um nível moderado.

O modelo de banco de dados *New Structured Query Language* (NewSQL) é baseado em uma junção dos principais paradigmas de banco de dados utilizado, ou seja, o SQL e o NoSQL com o avanço dessa nova tecnologia será possível realizar consultas de forma estruturada que é uma característica dos bancos de dados SQL. Com isso, poderá ser feito consultas respeitando as propriedades ACID, com a junção da velocidade e disponibilidade do paradigma NoSQL que permite salvar dados em memória volátil, deixando assim as operações de banco de dados muito mais rápidas. Logo que é possível distribuir os seus dados em diversos servidores gerando assim uma Linguagem estruturada de dados (DQL) muita mais ágil e ao mesmo tempo consistente.

Com a chegada da rede 5G e o novo *hardware* que o novo século disponibiliza esse tipo de junção se torna possível, pois, os antigos paradigmas de banco de dados foram baseados em uma arquitetura mais antiquada de *hardware* e de redes.

Com o novo paradigma NewSQL será possível usar as habilidades de ambos para fazer: Linguagem de definição de dados (DDL), Linguagem de manipulação de dados (DML), (DQL) Linguagem de transação de dados (DTL) e Linguagem de controle de dados (DCL), em grande banco de dados que rodam em redes Processamento de Transações em Tempo Real (OLTP) e em outros tipos de redes com alta carga de dados.

Por fim, o objetivo deste trabalho é analisar as principais diferenças deste

paradigma e realizar um teste de *benchmarking* em um Sistema de Gerenciamento de Banco de Dados (SGBD) que tenha suporte a essa tecnologia.

## 2 FUNDAMENTAÇÃO TEÓRICA

### 2.1 NewSQL

Segundo Aslett, Matthew (2011) “O NewSQL é uma classe de sistemas de gerenciamento de banco de dados relacional que busca fornecer a escalabilidade dos sistemas *Not Only Structured Query Language* (NoSQL) para cargas de trabalho de processamento de transações on-line (OLTP), mantendo as garantias ACID de um sistema de banco de dados tradicional”

Segundo Aslett, Matthew (2011), o banco de dados NewSQL veio com esse novo pensamento que pretende dar muito certo no nosso mundo atual.

Segundo Grolinger (2013), define o modelo NewSQL como sendo baseado no modelo relacional, oferecendo uma visão puramente relacional dos dados. Embora os dados possam ser armazenados em forma de tabelas e relações, é interessante observar que as soluções NewSQL podem usar diferentes representações de dados.

Segundo Aslett e Hoff, “Muitos sistemas corporativos que lidam com dados de alto perfil (por exemplo, sistemas financeiros e de processamento de pedidos) são muito grandes para bancos de dados relacionais convencionais, mas têm requisitos de consistência e transação que não são práticos para sistemas NoSQL. Aslett, Matthew (2011), Hoff, Todd (2012)” As únicas opções disponíveis anteriormente para essas organizações eram a compra de computadores mais poderosos ou o desenvolvimento de *middleware* personalizado que distribui solicitações pelo DBMS convencional. Ambas as abordagens apresentam altos custos de infraestrutura e / ou custos de desenvolvimento. Os sistemas NewSQL tentam reconciliar os conflitos.

Por fim o objetivo principal do NewSQL é fazer com que evolua a forma como o banco de dados é tratado dando a possibilidade de utilizarmos todo o potencial que temos com as novas tecnologias que estamos utilizando atualmente como o 5G e arquitetura de servidores em nuvem que possibilitam alta escalabilidade para esse tipo de Sistema de Gerenciamento de Banco de Dados (SGBD).

### 2.2 Banco de dados VoltDB

O banco de dados VoltDB “alimenta negócios em tempo real em um mundo conectado. A plataforma fornece decisões precisas, tomadas em menos de 10 milissegundos, para influenciar diretamente a monetização no momento, evitar fraudes digitais e acelerar iniciativas de transformação digital.(VoltDB, 2020)” é o principal banco de dados quando o assunto é NewSQL criado em 2014 por Dr. Michael Stonebraker foi pensado para atingir o máximo potencial tecnológico oferecido pelo nosso século e ajudar principalmente em redes Processamento de Transações em Tempo Real (OLTP) e em transações de Processamento de Análises em Tempo Real (OLAP) que são transações que dependem muito da assertividade tecnologia para dar certo e não ter problemas maiores como erros de transações e falha de processamento.

O “banco de dados VoltDB é capaz de atingir um rendimento 45 vezes maior do que os produtos de banco de dados atuais. A arquitetura também permite que os bancos de dados VoltDB sejam redimensionados com facilidade, particionando as tabelas de

banco de dados, e procedimentos armazenados que acessam essas tabelas, em vários sites, ou partições, e uma ou mais máquinas para criar o banco de dados distribuído. Cada site trabalha de forma independente, com isso, cada transação pode ser executada até a conclusão, sem que ocorram bloqueios de registros individuais (VoltDB,2020).”

Para garantir a consistência do banco de dados, o *VoltDB* define cada procedimento armazenado como uma transação, sendo finalizada ou revertida como um todo. Os dados e o processamento podem ser distribuídos a várias partições individuais, que cada uma contém uma parte exclusiva dos dados e do processamento (VoltDB, 2020)

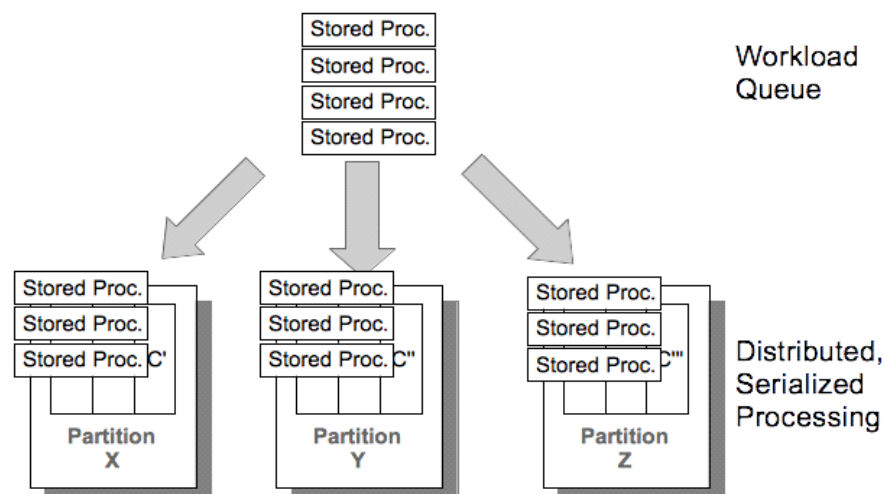
## 2.3 Características VoltDB

Entre as principais características do *VoltDB* estão a sua velocidade em lidar com transações simultâneas podendo retornar resultados em menos de segundos, resultados esses que um banco de dados estrutural não conseguiria fazer por limitações técnicas, diversas empresas confiam no *VoltDB* para cuidar de seus dados, segundo a *VoltDB*, 2020 “

Os recursos avançados do *VoltDB* ajudam você a obter mais valor dos seus investimentos em banco de dados, *data warehouse* e mensagens existentes” por isso desenvolvedores estão pensando duas vezes antes de contratar outro Sistema de Gerenciamento de Banco de Dados (SGBD) quando o assunto é criar sistemas Processamento de Análises em Tempo Real (OLAP). Segundo a *VoltDB*, 2020 estas são as características principais do seu banco de dados: “Gerenciamento de dados agregado, durável e escalável, geografia de transações, suporte a replicação, TCO gerenciável, integração de sistemas de filas, baixa latência consistente, nuvem nativa, lida com lógica complexa, qualidade de classe profissional e suporte e simplificação de pilha”.

O particionamento das tabelas, Figura 1, é feito baseado em coluna especificada, mas para conseguir otimizar ainda mais o desempenho, o *VoltDB* permite que tabelas pequenas, que sejam grande parte somente leitura, sejam replicadas para todas as partições do cluster, isso contribui para que a execução da transação seja de partição única (VoltDB, 2020).

**Figura 1 - Representação de particionamento do VoltDB**



Fonte: <https://www.voltdb.com> (2020)

## 2.4 Benchmarking

Um *benchmark* consiste em um padrão de testes para medir ou avaliar algo, sendo utilizado também para analisar o desempenho de um sistema. Pode ser implementado como uma instrução, um programa ou uma especificação de pedidos de interações com um componente de *software*” Lorena, 2007.

Segundo Svobodova (1976), “a finalidade da avaliação de desempenho de um Sistema e verificar a efetividade com que os recursos de *hardware* são utilizados para atender aos objetivos dos *softwares*. Esta avaliação pode ser classificada como comparativa, quando o desempenho do sistema e avaliado mediante o desempenho de outro sistema, ou analítica, quando o sistema e avaliado com vários parâmetros de carga e de configuração do próprio sistema.

Segundo Gray (1993), um *benchmark* deve atender os seguintes critérios:

a) Relevância: As medidas verificadas deverão descrever funcionalidade e expectativa de desempenho acerca do produto submetido ao *benchmark*. E necessário especificar com clareza o domínio da utilização da ferramenta e as operações a serem submetidas;

b) Portabilidade: Na elaboração de um *benchmark*, as definições e os termos utilizados devem estar em um nível de abstração que permita transportá-lo para as implementações de diferentes ferramentas. Portanto, não deve haver privilégio na utilização de um contexto que seja particular a uma determinada abordagem.

c) objetivo desta consideração e tornar o *benchmark* aplicável a uma gama maior de ferramentas a serem avaliadas;

d) Escalabilidade: O padrão das medições deve atender a pequenos ou grandes sistemas, independentemente do nível de complexidade dos mesmos fatores como monoprocessamento ou multiprocessamento, *clock* e tamanhos de memórias não devem tornar as medidas inconsistentes, nem influenciar na aplicabilidade do *benchmark*;

e) Simplicidade: Os critérios definidos no *benchmark* devem ser simples e de fácil compreensão. Por em a relevância das métricas não deve ser negligenciada. Medidas muito simples podem não refletir boas características de julgamento de uma ferramenta. No entanto, se for definido um conjunto de métricas muito complexas e de difícil assimilação pela comunidade interessada nos resultados, poderá ocorrer uma perda de credibilidade do *benchmark*.

## 3 MATERIAIS E MÉTODOS

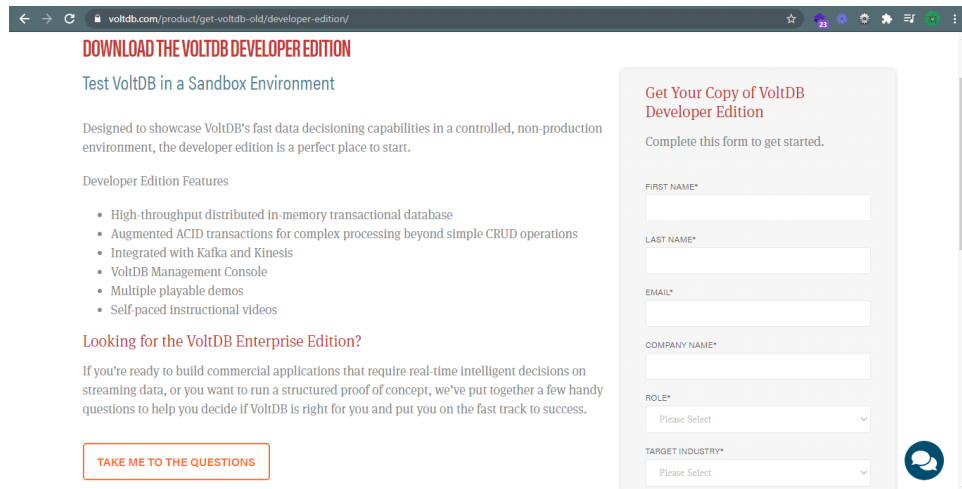
Como material para esse projeto foi utilizado as documentações do *SGBD VoltDB* (Segundo, *VoltDB*, 2011) que fornece todo o necessário para fazer a instalação do software e manipulação e em seguida realizar um teste de *benchmarking*. Em conjunto foi utilizado o banco de dados MySQL e uma aplicação em JAVA similar a utilizada pelo VoltDB, foi efetuado um teste sobre essas circunstancias onde foi obtido os resultados para realizar uma comparação entre os benchmarkings.

Os equipamentos utilizados para o projeto, foi um Notebook Itautec Windows 10 professional edition com as seguintes características de hardware: Intel(R) Core (TM) I3-2440M CPU @ 2.20GHz, 2200 Mhz, 2 Núcleo(s), 4 Processador(es) Lógico(s) Memória Física (RAM) Instalada 8,0 GB e um servidor cloud hospedado na Digital Ocean, onde os serviços foram instalados o servidor contém as seguintes características de hardware: 80GB de memória SSD, 4 GB de memória RAM, Ubuntu 18.04 (LTS) x64 e Intel(R) Core (TM) I3.

Para realizar o teste foi usado o *VoltDB Developer Edition* com a sua versão 10.1.1,

um sistema *Ubuntu 18.04* (caso esteja usando Windows pode-se usar o subsistema *Linux* que está disponível para o Windows 10) deve-se instalar uma *JVM (Java virtual machine)* no sistema *Linux* para execução das aplicações que irá executar os testes de *benchmark*. Para o teste com o MySQL foi utilizado a versão 14.14 do banco de dados.

**Figura 1 - Download VoltDB Developer Edition**

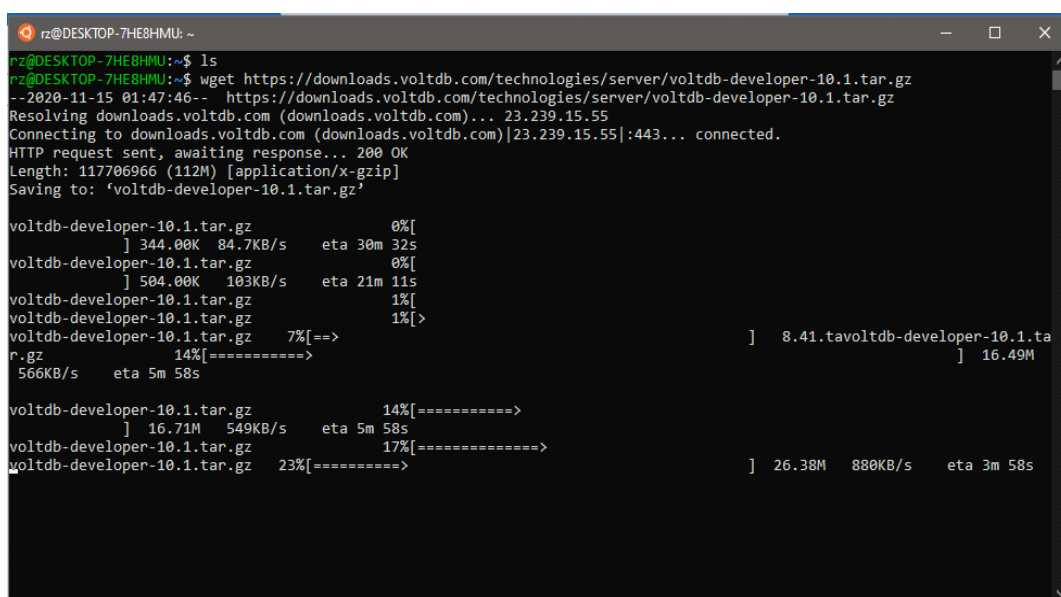


FONTE: <https://www.voltodb.com/product/get-voltdb-old/developer-edition/> (2020)

## 4 RESULTADOS E DISCUSSÃO

Para realizar o teste usa-se um exemplo já disponível na documentação do *VoltDB*, foi instalado o banco de dados, utilizando o comando *wget* como demonstrado na figura 2

**Figura 2 – Download VoltDB**

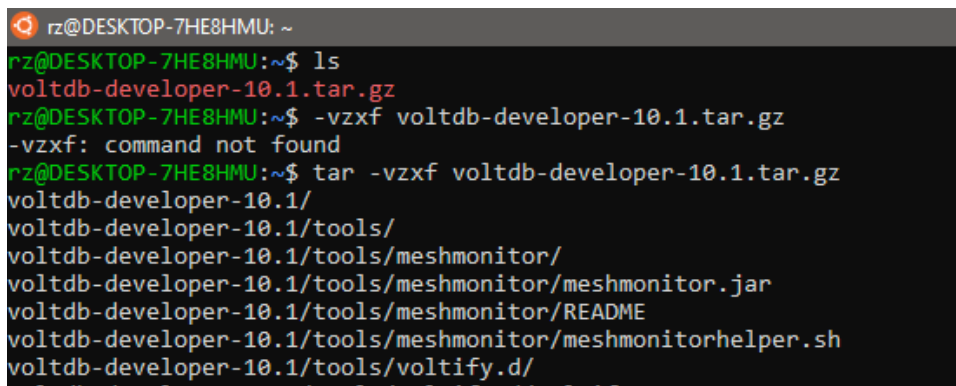


FONTE: Autor (2020)

Em seguida se faz necessário realizar a descompactação do banco de dados como

demonstra a figura 3 com o comando `tar -vzxf voltdb-developer-10.1.1.tar.gz`.

**Figura 3 – Descompactar**



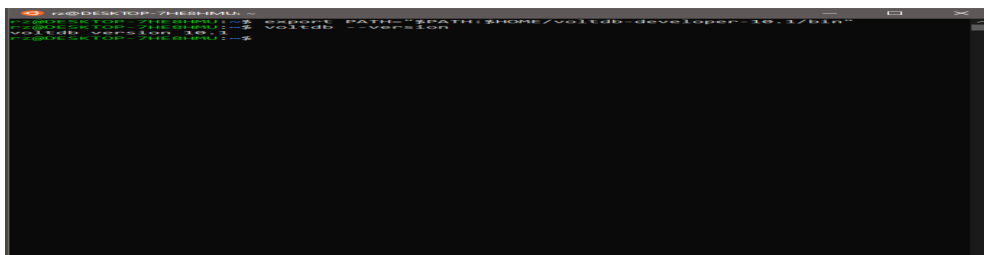
```
rz@DESKTOP-7HE8HMU: ~  
rz@DESKTOP-7HE8HMU:~$ ls  
voltdb-developer-10.1.tar.gz  
rz@DESKTOP-7HE8HMU:~$ -vzxf voltdb-developer-10.1.tar.gz  
-vzxf: command not found  
rz@DESKTOP-7HE8HMU:~$ tar -vzxf voltdb-developer-10.1.tar.gz  
voltdb-developer-10.1/  
voltdb-developer-10.1/tools/  
voltdb-developer-10.1/tools/meshmonitor/  
voltdb-developer-10.1/tools/meshmonitor/meshmonitor.jar  
voltdb-developer-10.1/tools/meshmonitor/README  
voltdb-developer-10.1/tools/meshmonitor/meshmonitorhelper.sh  
voltdb-developer-10.1/tools/voltify.d/
```

FONTE: Autor

(2020)

Para o correto funcionamento foi colocado o *VoltDB* em uma variável de caminho do *Linux* para isso rodar o comando `export PATH="$PATH:$HOME/voltdb-developer-10.1/bin"` foi testado o comando `voltdb -version` como na figura 4.

**Figura 4 – Variável ambiente**



FONTE: Autor (2020)

Após, criar um diretório com o comando `mkdir benchmark` onde será inicializado a instância do *VoltDB* em seguida faz-se necessário entrar no diretório com `cd benchmark` e se inicia uma nova instância do banco utilizando o comando `voltdb init` e assim foi colocado essa instância online com o comando `voltdb start` igual na figura 5. para continuar o teste foi deixado uma janela do terminal aberta rodando o servidor do banco de dados enquanto outra vai ser aberta para rodar a parte da aplicação.

**Figura 5 – Iniciando banco de dados**

```
rz@DESKTOP-7HE8HMU: /mnt/c/Users/razie$ mkdir benchmark
rz@DESKTOP-7HE8HMU: /mnt/c/Users/razie$ cd benchmark
rz@DESKTOP-7HE8HMU: /mnt/c/Users/razie/benchmark$ voltdb init
OpenJDK 64-Bit Server VM warning: Option UseConcMarkSweepGC was deprecate
ed in version 9.0 and will likely be removed in a future release.
Initializing VoltDB...

-----

Build: 10.1 voltdb-10.1-0-gab62e0f-local Community Edition
When using the INIT command, some deployment file settings (hostcount an
d voltdbroot path) are ignored
Initialized VoltDB root directory voltdbroot
rz@DESKTOP-7HE8HMU: /mnt/c/Users/razie/benchmark$ voltdb start
WARNING: Unsupported release: Ubuntu release 20.04 focal
OpenJDK 64-Bit Server VM warning: Option UseConcMarkSweepGC was deprecate
ed in version 9.0 and will likely be removed in a future release.
Initializing VoltDB...

-----

Build: 10.1 voltdb-10.1-0-gab62e0f-local Community Edition
Loaded node-specific settings from voltdbroot/config/path.properties
```

FONTE: Autor (2020)

O arquivo *ddl.sql* é responsável pela criação de tabelas no banco de dados com dois campos e um procedimento para fazer inserção de dados dentro da mesma como mostra a figura 6,

**Figura 6 – Executando arquivos**



```
rz@DESKTOP-7HE8HMU: - x rz@DESKTOP-7HE8HMU: / x + v - □ x
rz@DESKTOP-7HE8HMU:~/voltdb-developer-10.1/examples/simple$ cd ~/voltdb-developer-10.1/examples/simple/
rz@DESKTOP-7HE8HMU:~/voltdb-developer-10.1/examples/simple$ sqlcmd < ddl.sql

LOAD CLASSES procedures/procedures.jar;
Command succeeded.

FILE -inlinebatch END_OF_BATCH

-- simple example table
CREATE TABLE app_session (
  appid          INTEGER      NOT NULL,
  deviceid       BIGINT       NOT NULL,
  ts             TIMESTAMP    DEFAULT NOW
);
-- partitioning this table will make it fast and scalable
PARTITION TABLE app_session ON COLUMN deviceid;
-- create an index to allow faster access to the table based on a given deviceid
CREATE INDEX app_session_idx ON app_session (deviceid);

-- this view summarizes how many sessions have been inserted for each app / device combination
CREATE VIEW app_usage AS
SELECT appid, deviceid, count(*) as ct
FROM app_session
GROUP BY appid, deviceid;

-- you can declare any SQL statement as a procedure
CREATE PROCEDURE apps_by_unique_devices AS
SELECT appid, COUNT(deviceid) as unique_devices, SUM(ct) as total_sessions
```

FONTE: Autor (2020)

a partir daí foi executado outro arquivo chamado *run\_client.sh* que faz a compilação de uma classe *Java* que age como cliente no banco de dados fazendo a inserção dos dados através do procedimento criado anteriormente, criando visões para a tabela, atualizando os índices e salvando as alterações como mostra a figura 7.

**Figura 7 – Execução do procedimento**

```
root@rz: ~/voltdb-developer-10.1/examples/simple# ./run_client.sh localhost 500000
-----
Running Performance Benchmark for 500000 Transactions
-----
```

FONTE: Autor (2020)

A execução dos dois arquivos resultou no teste de *benchmark*, o parâmetro passado para o teste foi de inserir 500 mil linhas de dados, o teste foi executado 5x abaixo os resultados de cada execução:

**Figura 8 – Primeiro resultado do *Benchmark***

```
root@rz: ~/voltdb-developer-10.1.1/examples/simple# ./run_client.sh localhost 500000
-----
Running Performance Benchmark for 500000 Transactions
-----
Time      Txn/sec  Aborts  Failures
-----
00:00:05   37667    0        0
00:00:10   54672    0        0
-----

Benchmark Statistics
-----

Average throughput:                46,829 txns/sec

-----

Transaction Results
-----

insert_session
  calls: 500000
  commits: 500000
  rollbacks: 0

root@rz:~/voltdb-developer-10.1.1/examples/simple#
```

FONTE: Autor (2020)

**Figura 9 – Segundo resultado do *Benchmark***

```
root@rz: ~/voltdb-developer-10.1.1/examples/simple# ./run_client.sh localhost 500000
-----
Running Performance Benchmark for 500000 Transactions
-----
Time      Txn/sec  Aborts  Failures
-----
00:00:05   43555    0        0
00:00:10   47768    0        0
-----

Benchmark Statistics
-----

Average throughput:                46,820 txns/sec

-----

Transaction Results
-----

insert_session
  calls: 500000
  commits: 500000
  rollbacks: 0

root@rz:~/voltdb-developer-10.1.1/examples/simple#
```

FONTE: Autor (2020)

**Figura 10 – Terceiro resultado do *Benchmark***

```
root@rz: ~/voltdb-developer-11 x + v
root@rz:~/voltdb-developer-10.1.1/examples/simple# ./run_client.sh localhost 500000
-----
Running Performance Benchmark for 500000 Transactions
-----
Time      Txn/sec Aborts Failures
-----
00:00:05  41664   0      0
00:00:10  54887   0      0
-----

Benchmark Statistics
-----

Average throughput:                48,524 txns/sec

-----

Transaction Results
-----

insert_session
  calls: 500000
  commits: 500000
  rollbacks: 0

root@rz:~/voltdb-developer-10.1.1/examples/simple#
```

FONTE: Autor (2020)

**Figura 11 – Quarto resultado do *Benchmark***

```
root@rz: ~/voltdb-developer-11 x + v
root@rz:~/voltdb-developer-10.1.1/examples/simple# ./run_client.sh localhost 500000
-----
Running Performance Benchmark for 500000 Transactions
-----
Time      Txn/sec Aborts Failures
-----
00:00:05  39463   0      0
00:00:10  52506   0      0
-----

Benchmark Statistics
-----

Average throughput:                46,408 txns/sec

-----

Transaction Results
-----

insert_session
  calls: 500000
  commits: 500000
  rollbacks: 0

root@rz:~/voltdb-developer-10.1.1/examples/simple#
```

FONTE: Autor (2020)

**Figura 12 – Quinto resultado do *Benchmark***

```
root@rz: ~/voltdb-developer-10.1.1/examples/simple# ./run_client.sh localhost 500000
-----
Running Performance Benchmark for 500000 Transactions
-----
Time      Txn/sec  Aborts  Failures
-----
00:00:05  47995    0       0
-----
Benchmark Statistics
-----
Average throughput:          51,663 txns/sec
-----
Transaction Results
-----
insert_session
  calls: 500000
  commits: 500000
  rollbacks: 0
root@rz:~/voltdb-developer-10.1.1/examples/simple#
```

FONTE: Autor (2020)

Em seguida, foi feita a instalação do MySQL no Ubuntu e foi criado um banco de dados como sugere a figura 13. para manter o teste de benchmark justo entre os dois bancos foi criada a mesma tabela com os mesmos princípios de tipos de dados, índices, campos e visões, uma das vantagens que foi percebida é a retro compatibilidade da sintaxe do código entre as duas plataformas, sendo de fácil migração o código de um paradigma para o outro.

**Figura 13 – SQL da criação de banco de dados do MySQL**

```

1  -- CRIANDO BANCO DE DADOS E TABELA
2  -- (BASEADO NA TABELA DO VOLTDB MESMAS COLUNAS E MESMO TIPO DE DADOS)
3  CREATE DATABASE benchmark_mysql;
4  USE benchmark_mysql;
5
6  CREATE TABLE app_session(
7      appid INTEGER primary key auto_increment,
8      deviceid BIGINT NOT NULL,
9      ts timestamp
10 );
11
12 -- CRIANDO INDICES DA TABELA
13 CREATE INDEX app_session_idx ON app_session (deviceid);
14 -- CRIANDO UMA VIEW DA TABELA
15 CREATE VIEW app_usage AS
16 SELECT appid, deviceid, count(*) as ct
17 FROM app_session
18 GROUP BY appid, deviceid;
19 delimiter //
20 -- PROCEDURE DE INSERT, A IDEIA É CHAMAR ESSA PROCEDURE
21 -- 500 MIL VEZES PELA APLICAÇÃO JAVA
22 -- ASSIM PODEMOS MENSURAR O TEMPO DE EXECUÇÃO DO MYSQL/VOLTDB
23 CREATE PROCEDURE insert_app()
24 BEGIN
25     INSERT INTO app_session (appid, deviceid, ts)
26     VALUES (DEFAULT, FLOOR(RAND() * 401) + 99999999, now());
27 COMMIT;
28 END//

```

FONTE: Autor (2020)

Após a conclusão da configuração do ambiente do MySQL, foi hora de criar o código responsável por fazer as chamadas ao MySQL e assim poder executar o benchmark utilizando o JAVA como sugere a figura 14

**Figura 14.1 – Código aplicação JAVA MySQL**

```

BenchmarkMySQL_TCC.java
1  import java.sql.*;
2  import java.text.SimpleDateFormat;
3  public class BenchmarkMySQL_TCC {
4      static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
5      static final String DB_URL = "jdbc:mysql://127.0.0.1:3306/benchmark_mysql?autoReconnect=true&useSSL=false";
6      static final String USER = "finley";
7      static final String PASS = "password";
8      public static void main(String[] args) {
9          Connection conn = null;
10         CallableStatement stmt = null;
11         try {
12             Class.forName("com.mysql.jdbc.Driver");
13             System.out.println("Conectando ao banco de dados: " + DB_URL);
14             conn = DriverManager.getConnection(DB_URL,USER,PASS);
15             System.out.println("Conectado!");
16             stmt = conn.prepareCall("{CALL insert_app()}");
17             Long started = 0;
18             Long end = 0;
19             Long difference = 0;
20             Long total = 0;
21             Long executions = 5000000;
22             SimpleDateFormat simpleDateFormat = new SimpleDateFormat("S.ss");
23             SimpleDateFormat totalFormat = new SimpleDateFormat("mm:ss:S");
24             System.out.println("Iniciando Benchmark para " + executions + " de chamadas a procedure");
25             for(Long i = 0; i <= executions; i++){
26                 started = System.currentTimeMillis();
27                 stmt.execute();
28                 end = System.currentTimeMillis();
29                 difference = end - started;
30                 total += difference;
31                 if(i == 10000){
32                     String time = simpleDateFormat.format(total);

```

FONTE: Autor (2020)

Figura 14.2 – Código aplicação JAVA MySQL

```
BenchmarkMySQL_TCC.java
31     if(i == 10000){
32         String time = SimpleDateFormat.format(total);
33         System.out.println(i + " chamadas a procedure Txn/Sec do intervalo entre a execução de cada uma = " + time);
34     }else if(i == 200000){
35         String time = SimpleDateFormat.format(total);
36         System.out.println(i + " chamadas a procedure Txn/Sec do intervalo entre a execução de cada uma = " + time);
37     }else if(i == 300000){
38         String time = SimpleDateFormat.format(total);
39         System.out.println(i + " chamadas a procedure Txn/Sec do intervalo entre a execução de cada uma = " + time);
40     }else if(i == 400000){
41         String time = SimpleDateFormat.format(total);
42         System.out.println(i + " chamadas a procedure Txn/Sec do intervalo entre a execução de cada uma = " + time);
43     }else if(i == 500000){
44         String time = SimpleDateFormat.format(total);
45         System.out.println(i + " chamadas a procedure Txn/Sec do intervalo entre a execução de cada uma = " + time);
46     }
47     }
48     String finalTime = totalFormat.format(total);
49     String averageTime = SimpleDateFormat.format(total / 5);
50     System.out.println("Media de intervalo de Txn/Sec a cada execução da procedure: " + (averageTime) );
51     System.out.println("Tempo total da execução do benchmark = " + (finalTime) );
52     stmt.close();
53     conn.close();
54     }catch(SQLException se){
55         se.printStackTrace();
56     }catch(Exception e){
57         e.printStackTrace();
58     }
59 }
60 }
```

FONTE: Autor (2020)

O teste de benchmark foi executado por 5 vezes assim como no VoltDB ambos os bancos de dados estavam com novas instancias e zerados antes dos testes, abaixo segue as imagens com os testes do MySQL.

Figura 15 – Primeiro resultado do benchmark

```
root@rz:~# javac BenchmarkMySQL_TCC.java
root@rz:~# java BenchmarkMySQL_TCC
Conectanco ao banco de dados: jdbc:mysql://127.0.0.1:3306/benchmark_mysql?autoReconnect=true&useSSL=false
Conectado!
Iniciando Benchmark para 500000 de chamadas a procedure
100000 chamadas a procedure Txn/Sec do intervalo entre a execução de cada uma = 69142
200000 chamadas a procedure Txn/Sec do intervalo entre a execução de cada uma = 89025
300000 chamadas a procedure Txn/Sec do intervalo entre a execução de cada uma = 13310
400000 chamadas a procedure Txn/Sec do intervalo entre a execução de cada uma = 18855
500000 chamadas a procedure Txn/Sec do intervalo entre a execução de cada uma = 11939
Media de intervalo de Txn/Sec a cada execução da procedure: 823,43
Tempo total da execução do benchmark = 03:39:119
```

FONTE: Autor (2020)

Figura 16 – Segundo resultado do benchmark

```
root@rz:~# java BenchmarkMySQL_TCC
Conectanco ao banco de dados: jdbc:mysql://127.0.0.1:3306/benchmark_mysql?autoReconnect=true&useSSL=false
Conectado!
Iniciando Benchmark para 500000 de chamadas a procedure
100000 chamadas a procedure Txn/Sec do intervalo entre a execução de cada uma = 77142
200000 chamadas a procedure Txn/Sec do intervalo entre a execução de cada uma = 68820
300000 chamadas a procedure Txn/Sec do intervalo entre a execução de cada uma = 45803
400000 chamadas a procedure Txn/Sec do intervalo entre a execução de cada uma = 27549
500000 chamadas a procedure Txn/Sec do intervalo entre a execução de cada uma = 98334
Media de intervalo de Txn/Sec a cada execução da procedure: 996,42
Tempo total da execução do benchmark = 03:34:983
```

FONTE: Autor (2020)

**Figura 17 – Terceiro resultado do benchmark**

```

root@rz:~# java BenchmarkMySQL_TCC
Conectanco ao banco de dados: jdbc:mysql://127.0.0.1:3306/benchmark_mysql?autoReconnect=true&useSSL=false
Conectado!
Iniciando Benchmark para 500000 de chamadas a procedure
100000 chamadas a procedure Txn/Sec do intervalo entre a execucao de cada uma = 40044
200000 chamadas a procedure Txn/Sec do intervalo entre a execucao de cada uma = 19824
300000 chamadas a procedure Txn/Sec do intervalo entre a execucao de cada uma = 96108
400000 chamadas a procedure Txn/Sec do intervalo entre a execucao de cada uma = 7954
500000 chamadas a procedure Txn/Sec do intervalo entre a execucao de cada uma = 91637
Media de intervalo de Txn/Sec a cada execucao da procedure: 583,43
Tempo total da execucao do benchmark = 03:37:916
root@rz:~#
    
```

FONTE: Autor (2020)

**Figura 18 – Quarto resultado do benchmark**

```

root@rz:~# java BenchmarkMySQL_TCC
Conectanco ao banco de dados: jdbc:mysql://127.0.0.1:3306/benchmark_mysql?autoReconnect=true&useSSL=false
Conectado!
Iniciando Benchmark para 500000 de chamadas a procedure
100000 chamadas a procedure Txn/Sec do intervalo entre a execucao de cada uma = 71442
200000 chamadas a procedure Txn/Sec do intervalo entre a execucao de cada uma = 17524
300000 chamadas a procedure Txn/Sec do intervalo entre a execucao de cada uma = 69304
400000 chamadas a procedure Txn/Sec do intervalo entre a execucao de cada uma = 36046
500000 chamadas a procedure Txn/Sec do intervalo entre a execucao de cada uma = 25929
Media de intervalo de Txn/Sec a cada execucao da procedure: 851,41
Tempo total da execucao do benchmark = 03:29:259
root@rz:~#
    
```

FONTE: Autor (2020)

**Figura 19 – Quinto resultado do benchmark**

```

root@rz:~# java BenchmarkMySQL_TCC
Conectanco ao banco de dados: jdbc:mysql://127.0.0.1:3306/benchmark_mysql?autoReconnect=true&useSSL=false
Conectado!
Iniciando Benchmark para 500000 de chamadas a procedure
100000 chamadas a procedure Txn/Sec do intervalo entre a execucao de cada uma = 63744
200000 chamadas a procedure Txn/Sec do intervalo entre a execucao de cada uma = 15329
300000 chamadas a procedure Txn/Sec do intervalo entre a execucao de cada uma = 80112
400000 chamadas a procedure Txn/Sec do intervalo entre a execucao de cada uma = 41354
500000 chamadas a procedure Txn/Sec do intervalo entre a execucao de cada uma = 81935
Media de intervalo de Txn/Sec a cada execucao da procedure: 163,43
Tempo total da execucao do benchmark = 03:35:819
    
```

FONTE: Autor (2020)

**Figura 20 – Tabela comparativa resultados benchmark**

VOLTDDB	MYSQL
1 - execucao = 10seg	1 - execucao = 3 minutos e 39 segundos
2 - execucao = 10seg	2- execucao = 3 minutos e 34 segundos
3 - execucao = 10seg	3- execucao = 3 minutos e 37 segundos
4 - execucao = 10seg	4- execucao = 3 minutos e 29 segundos
5- execucao = 5seg	5- execucao = 3 minutos e 35 segundos

FONTE: Autor (2020)

## 5 CONCLUSÃO

De acordo com a tabela comparativa figura 20 Foi concluído que o banco de dados *VoltDB* tem uma facilidade de lidar com grandes volumes de dados sendo inseridos ao

mesmo tempo e com maestria consegue lidar com o processamento, deixando um equilíbrio entre desempenho e qualidade.

Podendo ser usado para analisar redes que precisem de um processamento de dados contínuo como bancos e bolsa de valores, a estrutura do *NewSQL* foi pensada para realizar o tipo de tarefa que o 5G irá requisitar. Contudo, em um mundo cada vez mais tecnológico o processamento ágil de dados é necessário para empresas, por fim é visto que isso poderá impactar no orçamento da empresa e fidelidade com os seus clientes.

## 6 TRABALHOS FUTUROS

Para trabalhos futuros é pretendido fazer comparações com outros sistemas de banco de dados, em diversas situações de hardware e paradigmas para assim descobrir o melhor desempenho entre eles.

## 7 REFERÊNCIAS

ASLETT, Matthew (2011). **"How Will The Database Incumbents Respond To NoSQL And NewSQL?"**. 451 Group (published April 4, 2011). Acesso em 04 de mar. de 2020. Disponível em: <http://cs.brown.edu/courses/cs227/archives/2012/papers/newsq/aslett-newsq.pdf>

ASLETT, Matthew e PAVLO, Andrew, (2016). **"What's Really New with NewSQL?"** (PDF). SIGMOD Record. Acesso 04 de mar. de 2020. Disponível em: <https://db.cs.cmu.edu/papers/2016/pavlo-newsq-sigmodrec2016.pdf>

ASLETT, Matthew, (April 6, 2011). **"What we talk about when we talk about NewSQL"**. 451 Group Acesso 04 de mar. de 2020. Disponível em: [https://blogs.451research.com/information\\_management/2011/04/06/what-we-talk-about-when-we-talk-about-newsq/](https://blogs.451research.com/information_management/2011/04/06/what-we-talk-about-when-we-talk-about-newsq/)

GUIMARAES, Lorena. **Um Aplicativo para Teste de Performances de Bancos de Dados Através dos Benchmarks TPC-C e TPC-HL**, 2007. Acesso em 06 de jul. de 2020 Disponível em: <http://www.lia.ufc.br/site.GRAY>,

J., **Database and Transaction Processing Performance Handbook**, (1993). Chapter 1. Morgan Kaufman Publishers. Disponível em: <http://jimgray.azurewebsites.net/BenchmarkHandbook/chapter1.pdf>

GROLINGER, Katarina et al. **Data management in cloud environments: NoSQL and NewSQL data stores**: Journal Of Cloud Computing: Advances, Systems And Applications, 2013. Disponível em: <https://link.springer.com/content/pdf/10.1186%2F2192-113X-2-22.pdf>. Acesso em 06 de jul. de 2020.

HOFF, Todd, (September 24, 2012). **"Google Spanner's Most Surprising Revelation: NoSQL is Out and NewSQL is In"**. Acesso 04 de mar. de 2020. Disponível em: <http://highscalability.com/blog/2012/9/24/google-spanners-most-surprising-revelation-nosql-is-out-and.html>

LLOYD, Alex,(2012). **"Building Spanner"** (PDF). Berlin Buzzwords (published June 5, 2012). Acesso 04 de mar. de 2020. Disponível em: [https://2012.berlinbuzzwords.de/sites/2012.berlinbuzzwords.de/files/slides/alex\\_loyd\\_keynote\\_bbuzz\\_2012.pdf](https://2012.berlinbuzzwords.de/sites/2012.berlinbuzzwords.de/files/slides/alex_loyd_keynote_bbuzz_2012.pdf)

STONEBRAKER, Michael, (June 16, 2011). **"NewSQL: An Alternative to NoSQL and Old SQL for New OLTP Apps"**. Communications of the ACM Blog. Acesso 04 de mar. de 2020. Disponível em: <https://cacm.acm.org/blogs/blog-cacm/109710-new-sql-an->



[alternative-to-nosql-and-old-sql-for-new-oltp-apps/fulltext](#)

SVOBODOVA, L., **Computer Performance Measurement and Evaluation Methods Analysis and Applications**. Vol. 2. Disponível em:

<https://archive.org/details/computerperforma0000svob>

VolDB: **Developer Edition Download**, Página inicial. Acesso em: 14 de novembro. de 2020. Disponível em: <https://www.voltdb.com/product/get-voltdb-old/developer-edition/>.

VolDB: **GITHUB**, Página inicial. Acesso em: 14 de novembro. de 2020. Disponível em: <https://github.com/voltdb>

Código do autor, Página inicial. Acesso em: 14 de novembro. de 2020. Disponível em: <https://github.com/RazielMiranda/tcc-final-version/tree/main>