UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA E
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

RAFHAEL WESLLEY LEMES DE SOUZA

ANÁLISE DE DESEMPENHO DE BANCOS DE DADOS RELACIONAIS A PARTIR DA CONSTRUÇÃO DE UM SISTEMA DE *BENCHMARK* SINTÉTICO

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA 2020

RAFHAEL WESLLEY LEMES DE SOUZA

ANÁLISE DE DESEMPENHO DE BANCOS DE DADOS RELACIONAIS A PARTIR DA CONSTRUÇÃO DE UM SISTEMA DE *BENCHMARK* SINTÉTICO

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina Trabalho de Conclusão de Curso, do curso de Engenharia de Computação da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para a obtenção do título de Bacharel.

Orientador: Prof. Fabiano Scriptore de Carvalho.

CURITIBA 2020

RAFHAEL WESLLEY LEMES DE SOUZA

ANÁLISE DE DESEMPENHO DE BANCOS DE DADOS RELACIONAIS A PARTIR DA CONSTRUÇÃO DE UM SISTEMA DE BENCHMARK SINTÉTICO

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

Data de aprovação: 11 / setembro / 2020

_____

Anelise Munaretto Fonseca
Doutorado
Universidade Tecnológica Federal do Paraná

_____

Mauro Sérgio Pereira Fonseca
Doutorado
Universidade Tecnológica Federal do Paraná

_____

Fabiano Scriptore de Carvalho
Mestrado
Universidade Tecnológica Federal do Paraná

CURITIBA

2022

DEDICATÓRIA

Dedico este trabalho a Jesus, senhor e salvador da minha vida, ao meu pai e minha mãe, Marcio e Sueli.

AGRADECIMENTOS

Agradeço ao meu professor e orientador Fabiano Scriptore de Carvalho por me auxiliar na realização deste trabalho, as professoras Anelise Munaretto Fonseca e Keiko Veronica Ono Fonseca pela ajuda prestada no processo de submissão da proposta à comissão avaliadora, ao coordenador do curso de Engenharia de Computação, professor Bogdan Tomoyuki Nassu pelas instruções com relação a escrita do trabalho de conclusão de curso, ao corpo docente responsável pelo ensino do conhecimento teórico e técnico, professores Douglas Roberto Jakubiak, Humberto Remigio Gamba, Heitor Silverio Lopes, Ana Cristina Barreiras Kochem Vendramin, Joao Alberto Fabro, Cesar Augusto Tacla, Paulo Cezar Stadzisz, Ricardo Dutra da Silva, Gustavo Benvenutti Borba, Andre Schneider de Oliveira, Inacio Andruski Guimaraes, Wilson Horstmeyer Bogado, Juliano Mourao Vieira, Fabiana Pottker, Marcelo Mikosz Goncalves, Miguel Antonio Sovierzoski, Paulo Roberto Brero de Campos, Joao Israel Bernardo, Cezar Manuel Vargas Benitez, Hermes Irineu Del Monego, Raul Marques Pereira Friedmann, Leandro Batista de Almeida, Laudelino Cordeiro Bastos, Mauro Sergio Pereira Fonseca, Luiz Augusto Pelisson, Ricardo Luders, Gregorio Jedyn, Luiz Alberto Pavelsky da Costa e Paula Francis Benevides.

RESUMO

Os bancos de dados relacionais são estruturas de armazenamento de dados que trabalham em conjunto de outros *softwares* disponibilizando informações relevantes para a execução dos mesmos. Advindo de um conceito criado na década de 1970, o modelo relacional, esta tecnologia vem sendo aprimorada para alocar e recuperar informações de forma rápida e consistente. Em virtude deste fato, estruturas simples de armazenamento de dados se tornaram sistemas que gerenciam grandes quantidades de informações, realizando até mesmo cálculos com o conteúdo armazenado e disponibilizando resultados que ajudam nas decisões de negócio das empresas. Como existem diversos sistemas gerenciadores de bancos de dados com finalidades distintas, este trabalho realizou uma análise comparativa de sistemas da categoria OLTP, entregando como métrica comparativa o tempo que estes sistemas demoram para executar lotes de comandos SQL em um ambiente monousuário. O trabalho avaliou o desempenho destas ferramentas com diferentes quantidades de operações. A comparação foi realizada entre três *softwares*, MySQL, PostgreSQL e Firebird. Estes foram escolhidos utilizando um robô de pesquisas que avalia a preferência dessas ferramentas entre projetistas de *software*. Para a escolha dos gerenciadores, procurou-se mesclar ferramentas de alto e baixo conceito no mercado, para que fosse possível verificar se há diferenças significativas de desempenho entre os sistemas.

Palavras-Chave: *Banco de Dados, Análise de Desempenho, Análise Quantitativa*.

ABSTRACT

Relational databases are data storage structures that work together with other software, providing relevant information for their execution. Coming from a concept created in the 1970s, the relational model, this technology has been improved to allocate and retrieve information quickly and consistently. Due to this fact, simple data storage structures have become systems that manage large amounts of information, even performing calculations with the stored content and providing results that help companies' business decisions. As there are several database management systems for different purposes, this work carried out a comparative analysis of systems in the OLTP category, delivering as a comparative metric the time it takes these systems to execute batches of SQL commands in a single user environment. The work evaluated the performance of these tools with different amounts of operations. The comparison was made between three softwares, MySQL, PostgreSQL and Firebird. These were chosen using a research robot that assesses the preference of these tools among software designers. For the choice of managers, we tried to mix high and low concept tools in the market, so that it was possible to verify if there are significant differences in performance between the systems.

Keywords: Database, Performance Analysis, Quantitative Analysis.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACID	*Atomicity, Consistency, Isolation, Durability*
ANSI	*American National Standards Institute*
API	*Application Programming Interface*
ARO	*Army Research Office*
AS³AP	ANSI SQL *Standard and Portable*
CRUD	Create, Read, Update and Delete
DARPA	*Defense Advanced Research Projects Agency*
DCL	*Data Control Language*
DDL	*Data Definition Language*
DML	*Data Manipulation Language*
DQL	*Data Query Language*
DTL	*Data Transaction Language*
EDB	*Engineering Database Benchmark*
E/R	Entidade Relacionamento
IBM	*International Business Machines*
IDE	*Integrated Development Enviroment*
JDBC	Java Database Connectivity
JPA	Java Persistence API
JVM	Java Virtual Machine
NoSQL	*Not Only* SQL
NSF	*National Science Foundation*
OLAP	*On-Line Analytical Processing*
OLTP	*On-Line Transaction Processing*
ORM	Object Relational Mapping
PHP	*Hypertext Preprocessor*
SGBD	Sistema Gerenciador de Banco de Dados
SGBDR	Sistema Gerenciador de Banco de Relacional
SQL	*Structured Query Language*
TPC	*Transaction Processing Performance Council*
UML	*Unified Modeling Language*

SUMARIO

1. INTRODUÇÃO

De acordo com Heinemann Butterworth (2009), análise de desempenho ou *Benchmarking* é a comparação de produtos, serviços ou processos similares afim de identificar ou adaptar práticas que aprimorem suas capacidades.

O objeto do *Benchmarking* procura comparar suas características àquele semelhante que possui o melhor respaldo perante o mercado planejando sua pesquisa para ter retorno em seu investimento. Logo o resultado da análise leva a uma potencial fonte de ideias e informações que conduzem este a evoluir e atingir o alvo analisado (BONNICI, 2015).

De acordo com Bonnici (2015), as organizações engajam-se em investir recursos na análise comparativa para medir seguimentos de mercado como: desempenho estratégico, competitivo, financeiro, entre outros. O *Benchmarking* destes são decorrentes de uma metodologia aplicada que deve produzir resultados coerentes e proporcionar novos rumos estratégicos para a empresa.

Em ambientes computacionais, a análise comparativa ocorre pela execução de um *software* que objetiva realizar um conjunto restrito e pré-definido de operações para comparar dois ou mais sistemas informatizados. O *Benchmarking* computacional avalia diferentes tipos de *hardwares* ou *softwares* que executam a mesma função. Se os objetos avaliados são *softwares*, estes podem ser executados tanto em diferentes plataformas quanto em uma mesma (SENG, 2015).

O resultado retornado pela análise é uma métrica que tem a função de estabelecer a comparação entre dois ou mais sistemas por meio de informações técnicas.

Os *benchmarks* podem ser usados tanto para apresentar medidas qualitativas quanto quantitativas. Nos bancos de dados, a análise qualitativa avalia resultados como *backup* e recuperação de arquivos, interfaces que fazem a interação com o usuário facilitando assim o seu uso, retorno de informações de problemas do *software* ou ações do usuário, entre outros. A análise quantitativa compara o desempenho de duas ou mais bases de dados por meio da construção de experimentos controlados determinando aspectos como: velocidade, produtividade e capacidade (STRAWSER, 1984) (SENG, 2015).

Neste trabalhado é realizado uma análise quantitativa visando o parâmetro de saída em quantidade de operações por tempo em um ambiente monousuário

comparando os bancos de dados relacionais por meio das operações da linguagem *Structured Query Language* (SQL), linguagem a qual um banco de dados se comunica com outros sistemas, apresentando um *software* desenvolvido para esta finalidade.

O software desenvolvido executa a análise por meio de uma técnica chamada *Batch Processing* ou Processamento por Lotes de Dados. Com as operações de manipulação e busca de informações, ou seja, inserções, seleções, modificações e exclusões foram realizados testes com pequenos e grandes volumes de dados e com as operações da linguagem de definição da base de dados, criações e exclusões de tabelas, foram feitos testes com criações de uma, cem e mil tabelas.

Os bancos de dados sujeitos à análise foram escolhidos por meio de uma pesquisa realizada no site DB-Engines. Este cataloga os Sistemas Gerenciadores de Banco de Dados (SGDB) automaticamente a cada mês de acordo com os seguintes parâmetros: número de menções dos sistemas em *websites* e de empregos anunciados na Internet para trabalhar com uma determinada tecnologia.

Como esta ferramenta possui uma lista de bancos de dados tanto relacionais como não relacionais, foram escolhidos os bancos relacionais que são de código aberto e mantém taxas de atualizações e novas versões lançadas continuamente.

Para a escolha dos bancos de dados também levou-se em consideração o nível de relevância para o mercado que estas ferramentas possuem, procurando escolher *softwares* com altas e baixas qualificações. Constatou-se que a escolha do *software* MySQL, PostgreSQL e Firebird eram as opções viáveis para a realização do experimento.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Analisar o desempenho dos bancos de dados relacionais *open source* MySQL, PostgreSQL e Firebird, submetendo-os à mesma carga de trabalho e analisando os mesmos com diferentes operações *SQL.*

1.1.2 Objetivos Específicos

- Desenvolver aplicação que realize operações de inserção, busca, atualização e remoção sobre um banco de dados relacional;
- Desenvolver aplicação que realize as operações de criação e remoção de tabelas nas bases de dados dos bancos;
- Utilizar o sistema de criação e remoção de tabelas para gerar a mesma base de dados automaticamente em todos os bancos;
- Criar aplicação para geração de dados fictícios para preenchimento dos bancos de teste;
- Desenvolver um sistema de obtenção de métrica em quantidade de operações por tempo;
- Obter resultados dos bancos de dados submetendo-os a testes com as operações de inserção, seleção, atualização e exclusão;
- Obter resultados dos bancos de dados submetendo-os a testes com as operações de criação e remoção de tabelas;
- Analisar resultados sobre o desempenho de cada banco de dados diante das operações executadas.

1.2 JUSTIFICATIVA

Os bancos de dados MySQL e o PostgreSQL foram escolhidos em virtude de pesquisas realizadas a respeito da frequência de uso dessas ferramentas por desenvolvedores de *software* e analistas de bancos de dados.

O paradigma relacional compete com outros no quesito armazenamento de informação, estes são classificados como não relacionais e são divididos em várias categorias com diferentes abordagens (BRITO, 2010). Contudo os modelos relacionais são mais utilizados do que seus concorrentes (MACÁRIO e BALDO).

De acordo com Kristi Anderson (2019), o MySQL é o banco de dados de código aberto mais utilizado em projetos e aplicações, o PostgreSQL está em segundo lugar. Como esta pesquisa apresenta bancos de dados de diferentes paradigmas, é possível observar a importância que o paradigma relacional exerce sobre os sistemas

gerenciadores de bancos de dados, pois dois sistemas relacionais ocupam os primeiros lugares das preferências dos usuários.

O Firebird consta na lista do DB-Engines como o 32º sistema mais utilizado, logo ele foi escolhido por se tratar de um sistema que não é tão popular entre os desenvolvedores e analistas de bancos de dados.

Várias ferramentas para analisar o desempenho de Sistemas Gerenciadores de Banco de Dados Relacionais (SGBDR) foram desenvolvidas em um curto período de tempo, porém não existe uma ferramenta que analise todos os bancos de dados já produzidos, e sim um determinado número de sistemas que são mais conhecidos (SCALZO, 2018).

Existem poucas análises de desempenho com um *software* como o Firebird, também há a dificuldade em se encontrar projetos para fazer um analisador para um banco de dados como este. Logo, este trabalho justifica-se por apresentar uma análise de um sistema pouco conhecido em relação a bancos de dados populares do mercado, construindo um *software* para realizar essa tarefa.

A análise visa determinar as diferenças de desempenho deste sistema em relação a bancos de dados conhecidos no mercado, como o MySQL e o PostgreSQL, para isso construindo um *software* que realize as operações SQL nas bases de dados dos sistemas e obtenha uma métrica que caracterize a diferença entre eles.

A escolha do Java como linguagem de programação é devido a plataforma possuir *Application Programming Interfaces* (APIs) de conexão com bancos de dados que possuem muitas informações de como utiliza-las. *Sites* e fóruns de programação fornecem informações de como corrigir determinados erros que ocorrem durante o desenvolvimento do projeto, além disso, o Java é uma linguagem que é melhorada constantemente e isso torna o projeto algo que pode ser aprimorado sem maiores complexidades envolvendo as ferramentas em que o projeto foi desenvolvido (BRILL, 2002).

O Java Database Connectivity (JDBC) foi escolhido por meio de uma decisão que teve que ser tomada no início do desenvolvimento do projeto entre as diferentes APIs que a plataforma suporta, logo, foi optado pelo desenvolvimento do projeto com o JDBC pela sua facilidade com o manuseio das informações por meio de *queries* e também por ser a API padrão de comunicação entre uma aplicação Java e um banco de dados (LASCANO, 2008). Logo, existem muitas informações sobre como desenvolver projetos com essa API.

1.3 ESTRUTURA DO TRABALHO

Este trabalho está subdividido em seis capítulos, desenvolvendo gradativamente o tema a ser abordado:

- Introdução: capitulo introdutório com o objetivo geral, os objetivos específicos e a justificativa do trabalho;
- Fundamentação Teórica: apresentação dos conceitos necessários para o entendimento do projeto;
- Metodologia: sequência de estudos, simulações e construção do projeto;
- Resultados Obtidos: resultados dos testes realizados do conjunto aplicado na metodologia;
- Conclusões: análise final do experimento;
- Referencias: conteúdo teórico utilizado como base de estudo e escrita deste trabalho.

2. FUNDAMENTAÇÂO TEÓRICA

Este capítulo apresenta o conhecimento teórico a respeito dos bancos de dados relacionais assim como a metodologia pesquisada na elaboração do projeto.

2.1 BANCO DE DADOS

De acordo com Ramez Elmasri (1994) um banco ou base de dados refere-se a estruturas que possibilitam o armazenamento e a recuperação de informações relevantes a um sistema. Uma base de dados segue propriedades implícitas como:

- Representação de sua estrutura de armazenamento como um subconjunto do mundo real, ou seja, a tentativa de adequar o armazenamento de forma coerente e compreensível ao olhar humano;
- As coleções de informações dão significado ao sistema de forma que é possível saber sobre o propósito de uma determinada base de dados por meio da visualização de diagramas que refletem a mesma;
- A base de dados deve ser projetada para que atenda um sistema em sua totalidade no quesito armazenamento, para isso é necessário conhecer as características do mesmo.

Um banco de dados pode ser projetado e mantido por um conjunto de *softwares* escritos especificamente para esta tarefa ou por um Sistema Gerenciador de Banco de Dados (SGBD) (ELMASRI, 1994).

2.2 SISTEMAS GERENCIADORES DE BANCOS DE DADOS

Sistemas Gerenciadores de Bancos de Dados são *softwares* que promovem a interação entre usuários e a base de dados permitindo a aplicação de comandos no banco de dados como: criação, inserção, remoção e modificação, seleção, entre outros. Também são responsáveis por garantir a integridade e a segurança da informação, além de ter a função de recuperar as informações em caso de falha no sistema (RAMAKRISHNAN e GEHRKE, 2009).

Devido ao fato dos sistemas gerenciadores estarem diretamente envolvidos desde a criação até a população de informações na base de dados, ocorreu a união destes dois conceitos e atualmente quando refere-se a banco de dados, automaticamente inclui-se o próprio gerenciador, como se todo o conjunto composto pelo banco de dados e seus sistemas de interação com o usuário fossem uma coisa só (RAMAKRISHNAN e GEHRKE, 2009).

Um SGBD executa diversas tarefas que o tornam fundamental para o gerenciamento em uma base de dados como:

- Provimento de uma camada de abstração que permite disponibilizar os dados a um determinado programa que os demande sem que este tenha que manipular a base de dados diretamente, ou seja, somente o SGBD manipula tais informações tornando o acesso ao banco de dados mais seguro;
- Garantia da integridade dos dados por meio da checagem prévia dos métodos de manipulação do banco, caso o usuário tente acrescentar alguma informação incorreta a um determinado campo, o SGBD verifica o erro e interrompe imediatamente a requisição;
- Cuida de acessos concorrentes a uma determinada informação escalonando os usuários a um acesso por vez. Além disso, este garante que o próximo acesso terá os dados atualizados de acordo com a modificação ocorrida no acesso anterior;
- Facilita a compreensão do usuário nas operações do banco de dados fornecendo informações sobre erros de sintaxe cometidos ao fazê-las;
- Dispõe de recursos que possibilitam selecionar a autoridade de cada usuário, dependendo desta autoridade, alguns usuários poderão acessar todos os recursos do sistema e outros terão acessos limitados de leitura e escrita;
- Apresenta facilidade de recuperar falhas de *hardware* e *software* por meio da existência de arquivos de pré-imagem e sistemas de *backup* automáticos, demandando pouca interferência humana.

O SGBD compõe parte da estrutura de armazenamento que vigora nos sistemas atuais, na figura 1 é possível observar as estruturas que compõem um sistema gerenciador de banco de dados:

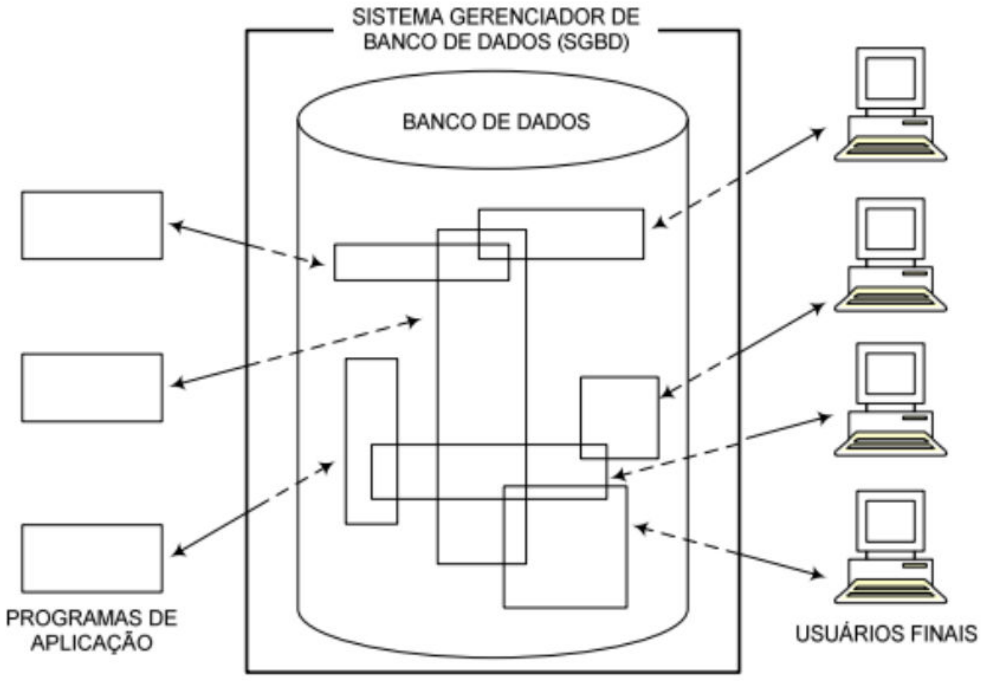


Figura 1 - Sistema de Gerenciador de Banco de Dados

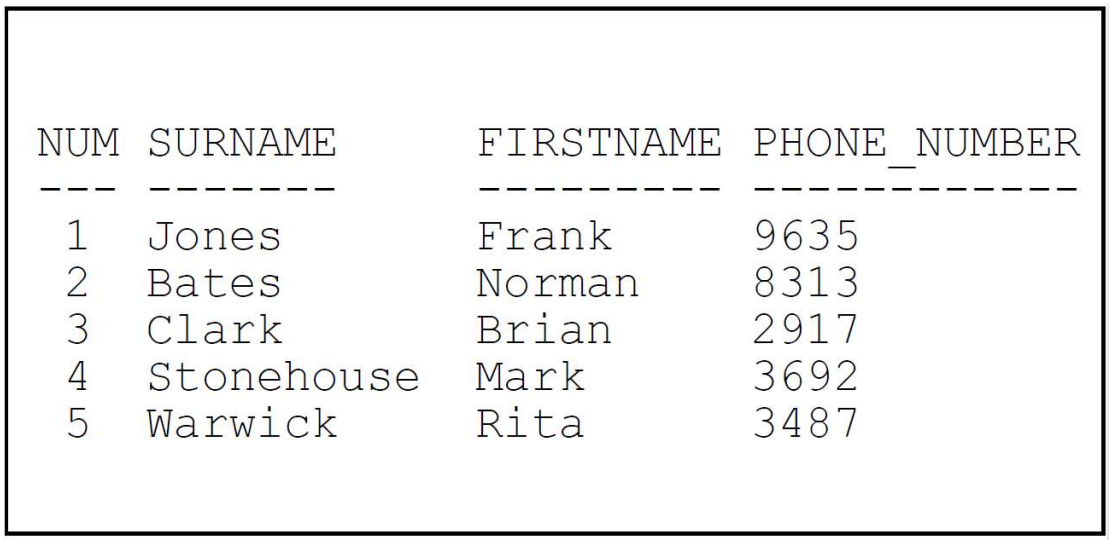
Fonte: (DATE, 2004)

2.3 BANCOS DE DADOS RELACIONAIS

2.3.1 Modelo Relacional

O modelo relacional foi proposto por Edgar Codd em 1970, como um novo paradigma para a representação de dados. A contribuição deste cientista em computação possibilitou avanços significativos na maneira como as informações são salvas nos bancos de dados (MOLINA, ULMAN e WIDOM, 1998).

De acordo com Akeel L. Din este modelo é extremamente utilizado devido ao fato de proceder de forma muito simples utilizando o conceito de “relação”, este nome é dado a uma tabela bidimensional que relaciona tipos de dados em comum conforme o exemplar mostrado na figura 2:



NUM	SURNAME	FIRSTNAME	PHONE_NUMBER
1	Jones	Frank	9635
2	Bates	Norman	8313
3	Clark	Brian	2917
4	Stonehouse	Mark	3692
5	Warwick	Rita	3487

Figura 2 - Exemplo de uma relação em banco de dados relacionais

Fonte: (DIN, 2012)

Tratando-se do modelo relacional, existem alguns conceitos que estão contidos em uma relação:

- Atributos: definem o significado das entradas em uma coluna, servem também como nomes para as entradas da mesma;
- *Schemas*: trata-se do nome e o conjunto de todos os atributos de uma relação. São representados por meio do nome da relação e a lista de atributos entre parêntesis. Ao exemplo da figura 2, denominando-se a relação como *LIST*, tem-se como a representação de um *schema* o formato *LIST*(*NUM*, *SURNAME*, *FIRSTNAME*, *PHONE_NUMBER*);
- Tuplas: também chamadas de *record*, são as linhas de uma relação que contém as informações de cada conjunto relacionado.

De acordo com Christopher J. Date (2004) as tabelas em um modelo relacional representam a estrutura lógica de armazenamento de dados e não sua estrutura física. Logo, os dados podem ser armazenados de várias formas fisicamente como: sequencial, indexação, *hashing,* entre outros. O que o modelo relacional proporciona é a estrutura de como estas informações serão visualizadas pelo usuário.

2.3.2 Linguagem SQL

A linguagem SQL é a forma com que um banco de dados relacional se comunica com outros sistemas. Ela contém os comandos de inserção, remoção, modificação e pesquisa de dados. Caracterizando as principais operações, para o conjunto destas denomina-se o acrônimo CRUD (*Create*, *Remove*, *Update*, *Delete*) (RUSSELL, 2015).

A linguagem SQL foi originada por meio de um projeto de pesquisa conduzido pela empresa IBM em meados dos anos de 1970, com o objetivo de verificar se as teorias do modelo relacional poderiam ser aplicadas a um produto comercial viável, ou seja, um SGDBR. A perspectiva a longo prazo destes investimentos mostrou que a linguagem SQL não somente viria a ser a fonte de comunicação entre os gerenciadores e outros sistemas como também revolucionaria a manipulação das bases de dados por meio de um procedimento não procedural (DIN, 2012).

As linguagens não procedurais reduzem a complexidade na execução de comandos. Sendo o SQL pertencente a esta categoria, permite ao usuário fazer operações somente se preocupando com o resultado da operação e não em especificar a maneira com que a linguagem deve proceder ao fazê-la. Logo, em uma operação de busca, por exemplo, o usuário do banco de dados basta apenas digitar o comando colocando como referência um atributo que deseja, não se preocupando em que posição da lista esta informação se encontra (DIN, 2012).

O formato de escrita de uma *query,* linha de comando SQL*,* pode sofrer pequenas modificações dependendo do gerenciador utilizado, mas em suma, há um padrão estabelecido para a escrita conforme a figura a seguir:

```
• insert into STUDENT (Name , Number, SchoolId)
  values ('John Smith', '100005', 1)

• select SchoolId, Name from SCHOOL

• select * from SCHOOL where SchoolId > 100

• update STUDENT set Name='John Wayne' where StudentId=2

• delete from STUDENT where SchoolId=3
```

Figura 3 - Operações CRUD

Fonte: (HALVORSEN, 2016)

A linguagem SQL é caracterizada por um padrão de escrita contendo uma série de elementos conforme a figura 4:

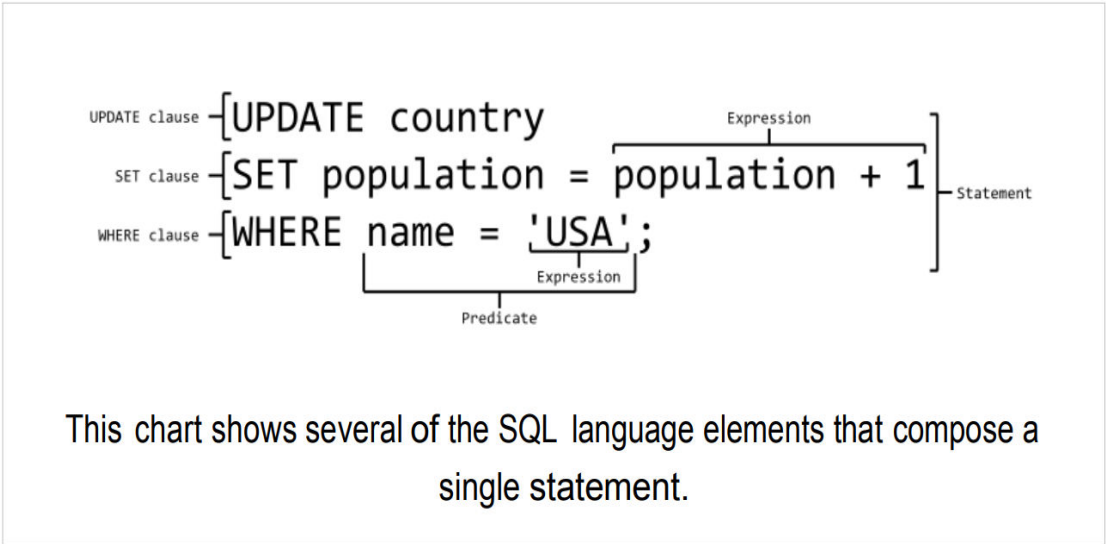


Figura 4 - Composição da Linguagem SQL

Fonte: (HALVORSEN, 2016)

- Cláusula (*Clause*): caracteriza o conjunto de comandos ou operações que a linguagem deve executar;
- Expressão (*Expression*): trata-se de parâmetros que determinam pontos específicos do que a operação deve fazer, quando utilizadas nos comandos SQL podem produzir valores escalares ou tabelas de dados;
- Predicados (*Predicates*): especificam condições que podem ser validadas na hora da execução do comando;
- Declaração (*Statement*): trata-se de todo o conjunto envolvendo as cláusulas, expressões e predicados.

A linguagem SQL é composta de sublinguagens, cada uma visa a atender uma necessidade do sistema com comandos divididos em categorias mostradas pela tabela 1:

Tabela 1 - Formato de Sublinguagens SQL

SQL	Definição	Explicação	Comandos
DDL	*Data Definition Language* - Linguagem de Definição de Dados.	comandos que interagem com os objetos do banco	CREATE
			ALTER
			DROP
DML	*Data Manipulation Language* - Linguagem de Manipulação de Dados	comandos que interagem com os dados dentro das tabelas	INSERT
			DELETE
			UPDATE
DQL	*Data Query Language* - Linguagem de Consulta de dados	comandos de consulta	SELECT
DTL	*Data Transaction Language* - Linguagem de Transação de Dados	comandos para controle de transação	BEGIN TRANSACTION
			COMMIT
			ROLLBACK
DCL	*Data Control Language* - Linguagem de Controle de Dados	comandos para controlar a parte de segurança do banco de dados	GRANT
			REVOKE
			DENY

Fonte: (ALVARO, 2016)

A partir das informações da tabela 1, percebe-se que as operações CRUD são provenientes de duas sublinguagens SQL, a linguagem DML e a DQL (ALVARO, 2016).

2.3.3 Entidade Relacionamento

De acordo com Christopher J. Date (2004), o modelo entidade relacionamento, ou modelo E/R, serve como referência para a construção de uma base de dados relacional. Também chamado de Modelo Semântico, visa proporcionar maior perspectiva às relações entre tabelas em um banco de dados apresentando uma representação gráfica de como estas informações estão interligadas.

O modelo trabalha com o conceito de entidade, uma representação de um objeto do mundo real que pode ser identificada de forma distinta, este objeto pode conter existência física, como por exemplo: uma pessoa, um carro, um prédio, entre outros, ou existência conceitual: um curso universitário, trabalho, empresa, etc.

Os relacionamentos referem-se às associações entre duas ou mais entidades. Existem várias formas de representar uma relação entre duas entidades em um diagrama E/R, o que torna este com muitas variantes, mas sempre mantendo a lógica

de sua representação (RAMAKRISHNAN e GEHRKE, 2009). A representação mais utilizada neste diagrama é por meio de símbolos nas linhas que ligam duas entidades, chamados de relações *Crows Foots*, estas podem ser vistas conforme a figura 5:

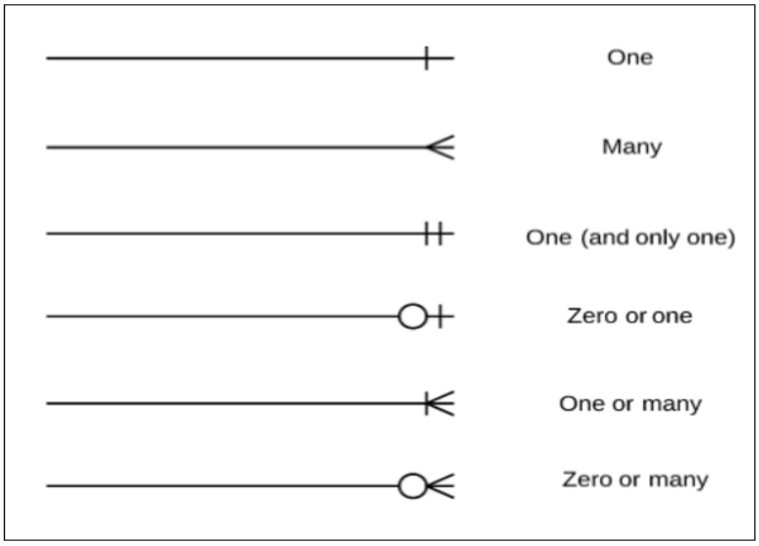


Figura 5 - Representação das Relações *Crows Foots*

Fonte: (LUCIDCHART, 2020)

Existem três tipos de relações entre entidades em um diagrama E/R: um para um (1x1), um para muitos (1xN) e muitos para muitos (NxM), cada uma possui uma ligação importante em sua construção. Estes relacionamentos podem ser vistos de acordo com a figura 6:

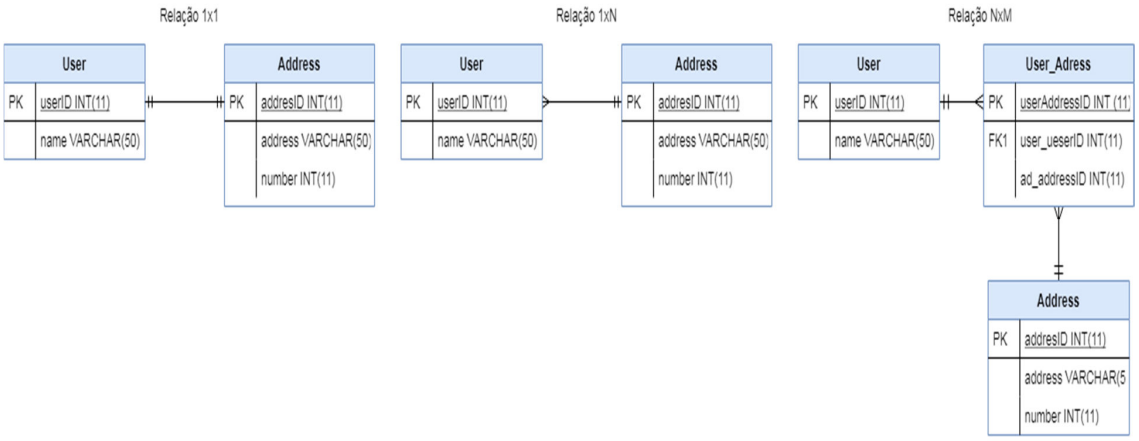


Figura 6 - Associações do modelo E/R pela representação Crows Foots

Fonte: Autoria Própria

De acordo com o exemplo da figura 6, é possível observar as relações entre as entidades no modelo E/R. As relações um para um caracterizam um objeto de uma tabela relacionando-se uma única vez com um objeto da outra, no exemplo pode-se notar que um usuário terá somente um endereço. Transpondo o mesmo exemplo para uma relação 1xN, nota-se que um endereço pode ter vários usuários, contudo cada usuário pode estar em apenas um endereço.

A relação NxM da figura 6 exemplifica a situação em que vários objetos de uma tabela possuem relação com vários objetos de outra, neste caso o exemplo mostra que além de um endereço poder ter mais de um usuário, vários usuários podem estar em um endereço.

2.3.4 ACID

Atomicidade, Consistência, Isolamento e Durabilidade são conjuntos de propriedades de transações de bancos de dados. Trata-se de um conceito que visa garantir a validade dos dados sobre qualquer problema que venha a ocorrer no sistema (RAMAKRISHNAN e GEHRKE, 2009).

Cada característica do acrônimo possui suas propriedades, sendo estas descritas como:

- Atomicidade: toda a transação em um banco de dados deve ser feita por completo ou falhar completamente, esta propriedade evita parcialidades em uma transação fazendo com que o usuário em um banco de dados não tenha dúvidas sobre o sucesso ou fracasso de sua tentativa ao executar um comando;
- Consistência: objetiva garantir que o processo da transação respeite todas as normas pré-estabelecidas na integridade dos dados, isto garante que o banco de dados será levado do começo ao fim da transação a um estado válido e evita informações corrompidas no armazenamento;
- Isolamento: garante que se uma operação for executada em um dado momento não será sobreposta por outras ao mesmo tempo, esta propriedade garante a veracidade dos dados;
- Durabilidade: garante que qualquer inserção ou alteração feita com sucesso deve permanecer no banco de dados até que seja realizada uma nova operação que venha a manipular este dado de forma a modifica-lo.

As propriedades ACID quando aplicadas a um banco de dados beneficiam não somente o sistema como também o desenvolvimento de aplicações que vão acessa-lo. Os desenvolvedores destas aplicações se preocuparão somente com a lógica das mesmas e não com fatores de segurança, recuperação e sincronização do acesso aos dados compartilhados.

Os bancos de dados relacionais sacrificam velocidade de execução para garantir as propriedades que foram discutidas.

2.3.5 Sistemas OLTP e OLAP

Os sistemas de processamento de transação, também conhecidos como *On-Line Transaction Processing* (OLTP), são bancos de dados relacionais desenvolvidos para suportar operações diárias de uma organização, este são caracterizados por um vasto número de pequenos comandos como: INSERT, DELETE, UPDATE e SELECT (STAIR e REYNOLDS, 2017).

O objetivo de um sistema OLTP é processar de forma rápida os comandos no banco de dados, mantendo a integridade das informações em acessos multiusuário e a consistência do banco.

Um sistema OLTP é caracterizado por ter um vasto número de usuários o utilizando, consequentemente gerando um ambiente de alta concorrência, tempos de resposta de operações curtos para manter a eficiência em um ambiente de acessos simultâneos, grande volume de dados e suporte para aguentar uma grande base de dados.

Os sistemas de processamento analítico, *ou On-Line Analytical Processing* (OLAP) são caracterizados por um baixo volume de transações e comandos complexos de manipulação de dados, estes comandos não constituem somente requisições feitas ao sistema, mas também são utilizados para produzir análises e resultados com informações contidas em uma base dados. Por exemplo, o usuário pode obter a informação do total de vendas realizadas em todos os departamentos de uma empresa multinacional nos últimos meses por meio de um único comando em um sistema OLAP (RAMAKRISHNAN e GEHRKE, 2009).

A tabela 2 contém informações detalhadas das diferenças entre os dois sistemas:

Tabela 2 - Diferenças entre Sistemas OLTP e OLAP

Característica	Base de Dados OLTP	Base de Dados OLAP
Proposito	Suporta processamento de transação de informações	Suporta tomada de decisão
Fonte de dados	Transações de Negócios	Múltiplos arquivos, bases de dados com informações internas e externas a empresa
Acesso de Dados Permitidos ao Usuário	Leitura e Escrita	Somente Leitura
Modelo primário de base de dados empregado	Relacional	Relacional
Nível de detalhamento	Transações Detalhadas	Dados frequentemente resumidos
Disponibilidade de dados históricos	Muito limitado - tipicamente algumas semanas ou meses	Muitos anos
Processo de Atualização	Online, conforme as transações são capturadas	Processo periódico, uma vez por semana ou por mês
Facilidade de processo	Rotineira e fácil	Complexa, deve combinar os dados de várias fontes, os dados precisam passar por um processo de verificação
Problemas de integridade de dados	Cada transação deve ser editada cuidadosamente	Esforços máximos para limpar e integrar os dados de múltiplas fontes

Fonte: (STAIR e REYNOLDS, 2017)

Os *benchmarks* são construídos para as categorias citadas anteriormente de forma que um sistema que avalie o resultado em uma categoria não pode ser usado estudar a outra (SADOGHI, BHATTACHERJEE e CANIM, 2018). Como muitos sistemas surgem constantemente, muitos *benchmarks* acabam sendo construídos com as características dos sistemas que precisam ser estudados, o que torna muitos estudos algo isolado (IVANOV1, RABL, *et al.*).

2.4 JAVA

De acordo com Scott Oaks e Henry Wong (1999), o Java define uma linguagem de programação, uma API para comunicação com diversos dispositivos de *software* e uma máquina virtual para a interpretação destes códigos. Logo, o Java é considerado como uma plataforma que provê vários serviços em diferentes segmentos da computação.

Os próximos tópicos tratam a respeito de componentes desta plataforma que possibilitaram o desenvolvimento do projeto.

2.4.1 JDBC

O JDBC é uma API para se estabelecer uma conexão entre um programa Java e uma base de dados relacional em um banco de dados (MARTIN, 2010).

Essa API possui uma série de bibliotecas que possibilitam o usuário criar e executar comandos SQL em um banco de dados a partir de um *software* escrito em Java. Além disso a API proporciona a modificação e obtenção de informações contidas em uma base de dados (SPEEGLE, 2001).

O JDBC utiliza-se de conectores para estabelecer a comunicação com o SGBD, cada sistema disponibiliza estes conectores a cada nova versão do seu *software*. Uma vez instalados os conectores em um projeto Java, a API reconhece os mesmos e passa a reportar erros conforme os retornos obtidos em tentativas ou perdas de conexão.

2.4.2 JPA

Java Persistence API assim como o JDBC tem a função de estabelecer uma conexão e manipular um banco de dados a partir de um programa escrito em Java, porém sobre um aspecto de transformação e conversão de informações e estruturas de bancos de dados relacionais em uma linguagem de programação orientada a objetos a partir de uma técnica conhecida como ORM (Object Relational Mapping) (FARIA e JUNIOR, 2015).

O ORM transforma entidades e objetos de um banco de dados em classes e variáveis de uma linguagem de programação. Contudo a necessidade do JPA em

empregar tal procedimento cria uma outra camada de código que usa a API do JDBC para a comunicação com o banco de dados, esta pode ser visualizada por meio da figura 7:

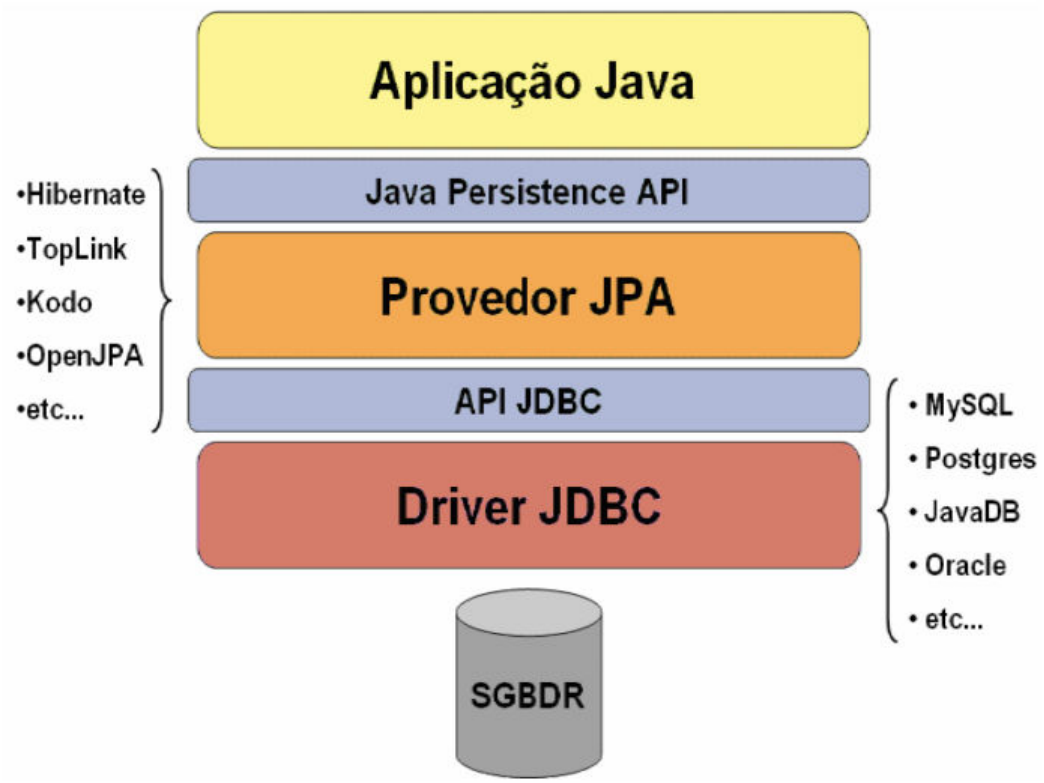


Figura 7 - Camadas de Código de Comunicação JDBC e JPA

Fonte: (OLIVEIRA, 2008)

De acordo com Jorge Edison Lascano (2016), para aplicações que são construídas com o objetivo de crescimento e precisam de uma estrutura de projeto que possa ser melhor desenvolvida e implementada, o uso do JPA é a melhor opção, pois oferece métodos de acesso e manipulação dos bancos de dados já implementados na API, além de proporcionar uma melhor visualização da aplicação por parte dos desenvolvedores uma vez que a base de dados pode ser interpretada por meio de classes Java. Contudo, para pequenas aplicações cujo objetivo é executar determinadas funções sem tomar muitos recursos do sistema a melhor opção para o desenvolvimento é com a API do JDBC, visto que o uso da API do JPA aumenta a complexidade do *software,* pois implementa o ORM.

2.4.3 JDBC Batch Processing

Batch Processing não é uma ferramenta exclusivamente da API do JDBC, mas é uma forma de processamento de dados em um banco (OCTAVIAN, RADULESCU, *et al.*, 2015). Este método permite que um lote de comandos SQL seja executado conforme a figura 8:

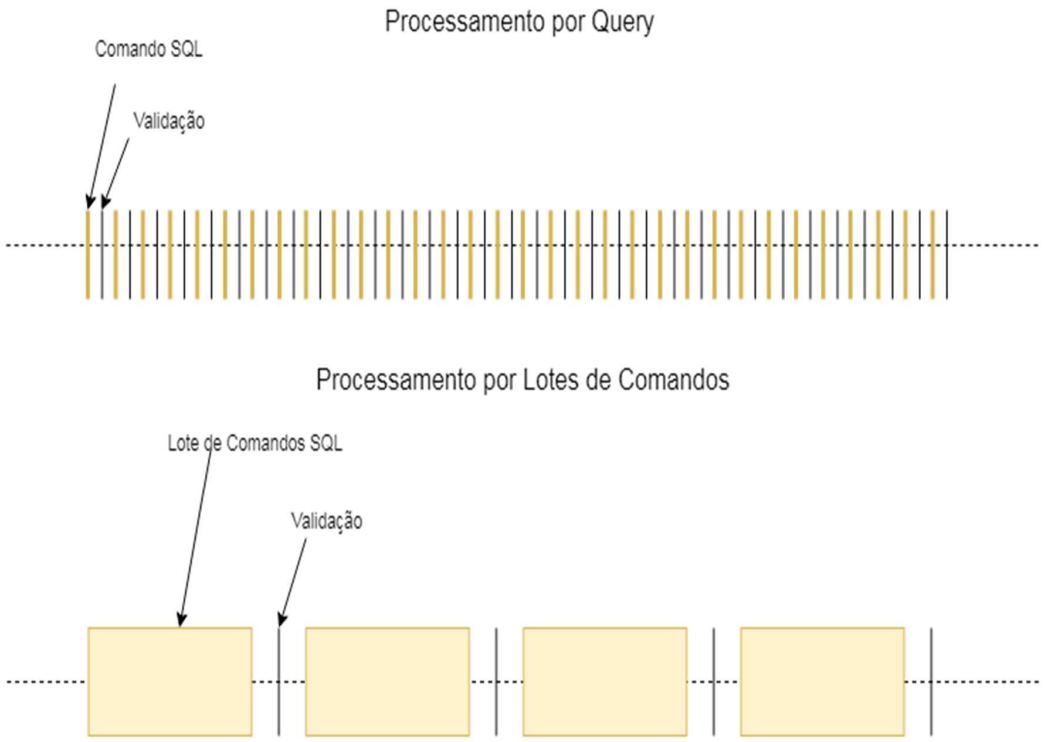


Figura 8 - Execução de Lotes de Comandos por *Batch Processing*

Fonte – Autoria Própria

Esta forma de processamento é executada pela API do JDBC por meio de uma função chamada Batch, esta função armazena vários comandos SQL, formando um lote de instruções, quando esse lote de comandos é enviado ao banco de dados, é realizado uma validação (BRILL, 2002).

Este método proporciona maior eficiência por parte do banco de dados no processamento das operações, pois a cada execução de uma única *query,* existem tempos gastos como: requisição de um outro *software* que necessita fazer uma operação em um banco de dados, o processamento desta requisição por parte do banco, validação da *query*, retorno do resultado, entre outros fatores (BRILL, 2002).

2.4.4 Java Virtual Machine

Bill Venners (1998) define uma JVM como sendo um interpretador de códigos escritos para a linguagem Java, foi desenvolvida para que a linguagem pudesse ser interpretada em diversos sistemas operacionais.

A forma como a JVM entrega um código interpretável a vários sistemas operacionais diferentes é por meio de um processo de conversão dos arquivos escritos para uma interpretação da própria máquina virtual, conforme a figura 9:

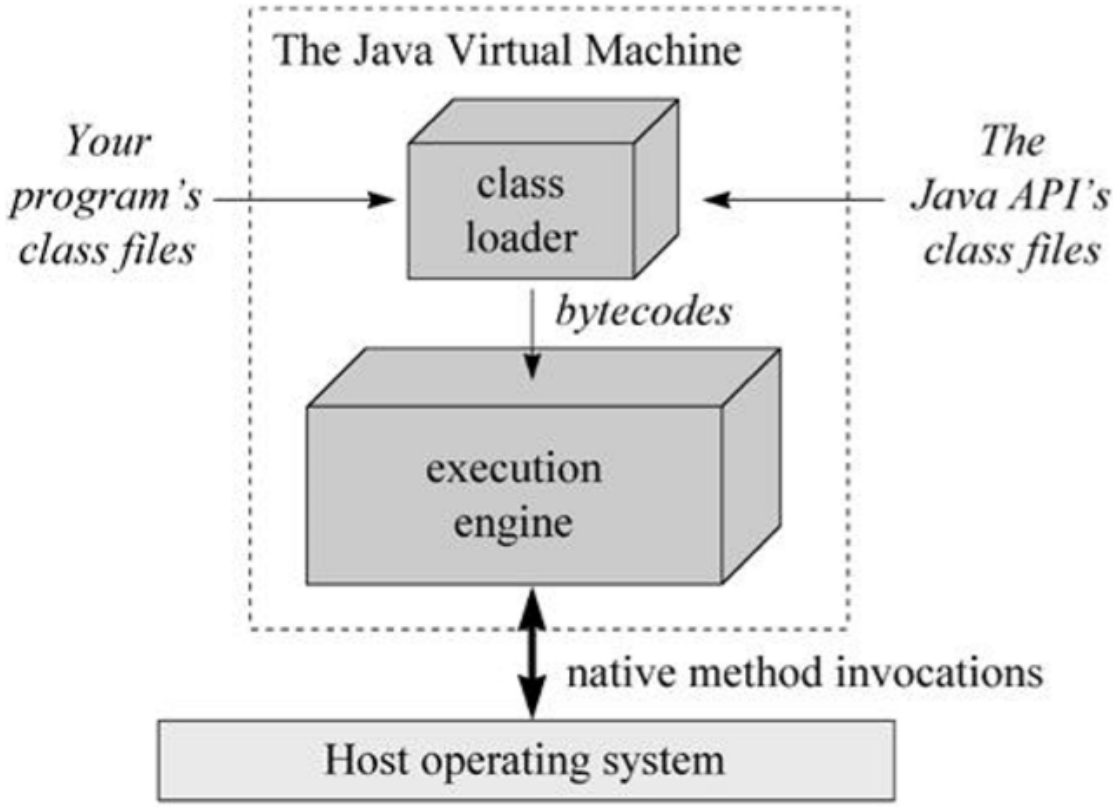


Figura 9 – Diagrama de Execução de uma Java Virtual Machine

Fonte: (VENNERS, 1998)

A figura 9 mostra o processo desde a interpretação da linguagem Java até a comunicação com um sistema operacional feita por uma JVM. A máquina virtual converte o código Java em algo interpretável por ela própria, chamados *bytecodes*.

Os *bytecodes* são executados por meio de um programa que possui funções escritas em linguagens como C, C++ e Assembly.

Estas funções são conhecidas como métodos nativos e categorizam diversas funcionalidades dos processadores em um computador, sendo assim utilizados pelo sistema operacional na hora da execução de suas tarefas.

A comunicação entre o sistema operacional e a máquina virtual é feita por meio destes métodos e desta forma um sistema pode interpretar um código escrito em Java mesmo que este tenha sido escrito em outro sistema operacional.

2.4.5 Java *Threads*

Threads são trechos de código de computador que são executados de forma independente, estas funções são executadas paralelamente para que o *software* possua maior velocidade para responder ao propósito ao qual foi criado (OAKS e WONG, 1999).

O termo remete a computação paralela, sendo uma solução criada para resolver problemas que ocorriam em execuções sequenciais, majoritariamente ligados a ineficiência desta solução no aproveitamento de recursos do *hardware* (GÖETZ, PEIERLS, *et al.*, 2006).

A solução envolvendo *threads* possibilitou um avanço significativo na execução de programas, os *softwares* passaram a aproveitar os tempos de ociosidade do sistema na espera da finalização de uma determinada tarefa ou parâmetros externos vindos dos usuários. Contudo, o sistema operacional passou a ter que lidar com trocas de contexto e acessos concorrentes de funções que executavam em paralelo, o termo logo ficou conhecido como *Multithreading* e significa a forma como o sistema operacional lida com as funções sem que uma interfira na outra.

Em Java, uma *thread* é executada estendendo uma classe Java chamada Thread ou implementando uma interface Java chamada Runnable Interface, a diferença entre elas é que estendendo uma classe Java Thread, a classe não poderá estender nenhuma outra, pois o Java não aceita múltiplas heranças, contudo, isto dá ao *software* uma estrutura de código simples e de fácil entendimento enquanto que a interface produz uma estrutura de código melhor e a possibilidade da implementação de várias outras interfaces ao mesmo tempo, entretanto, torna a estrutura do projeto complexa de se entender.

Threads em uma aplicação Java são executadas conforme mostra a figura 10:

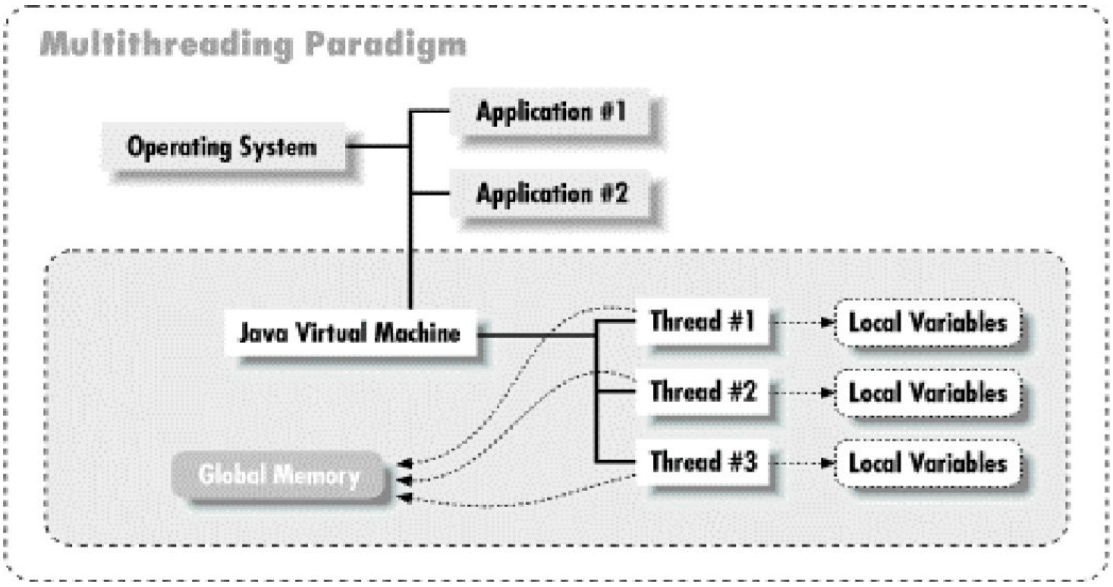


Figura 10 – Funcionamento do *Multithreading Java*

Fonte: (OAKS e WONG, 1999)

A partir da interpretação da figura 10, uma aplicação Java utiliza-se de uma JVM para interpretar as *threads* fazendo com que o sistema operacional possa entende-las e coordene suas execuções.

2.5 MYSQL

De acordo com Neil Smyth (2007), O MySQL é um sistema gerenciador de bancos de dados relacional (SGBDR), que utiliza a linguagem SQL para sua comunicação.

O MySQL foi criado na Suécia por Allan Larsson, David Axmark e Michael Widenus pela fundação de uma empresa chamada MySQL AB, esta disponibilizou a primeira versão do *software* em 1995.

Antes do surgimento do MySQL, as bases de dados eram muito complexas de serem criadas e seu custo era muito alto, as principais empresas que faziam a manutenção e a implementação do banco de dados eram as empresas Oracle e IBM (SMYTH, 2007).

O sucesso do MySQL é devido à fácil integração que este possui com linguagens de programação como o *Hypertext Preprocessor* (PHP) e Java, que são linguagens que proporcionam o desenvolvimento de aplicações pela Internet de forma

rápida com vários recursos previamente implementados, diminuindo assim o tempo de desenvolvimento de projetos, sendo um dos principais bancos de maior preferência por analistas de bancos de dados e desenvolvedores de *software*. Além disso o MySQL é confiável e seguro e é adotado como solução para a criação e gerenciamento de base de dados por muitas empresas.

De acordo com John Russell (2015), o MySQL está presente em diversas empresas como Google, Facebook, Youtube, Ebay e muitas outras, provendo suporte no armazenamento de dados e segurança de informação.

Segundo o manual referencial do MySQL (2020), este possui várias características importantes para os sistemas informatizados como:

- Portabilidade: pode ser executado em diversos sistemas operacionais o que possibilita grande vantagem no uso desta ferramenta;
- Compatibilidade: é possível fazer a interface do MySQL com outros *softwares* por meio de várias linguagens de programação e *frameworks* disponíveis para estas funções;
- *Design*: foi projetado para ser executado em modo *multithread*, várias tarefas executando ao mesmo tempo, quando o processador disponibiliza recursos para esta característica;
- Armazenamento: proporciona armazenamento em bases de dados transacionais e não transacionais;
- Busca de informação: utiliza-se de tabelas hash que funcionam como armazenamento temporário aumentando a velocidade da busca pelos dados e árvores binárias como algoritmo de organização lógica das tabelas em disco;
- Usabilidade: possibilidade de executar o sistema sem o uso de Internet pois este disponibiliza um servidor na própria máquina do usuário que pode ser acessado para testes e aplicações *standalone*;
- Conectividade: oferece a possibilidade não somente de usuários como também *softwares* clientes se conectarem ao banco.

2.6 POSTGRESQL

Assim como o MySQL, o PostgreSQL é um sistema gerenciador de banco de dados relacional que teve sua inicialização com o projeto POSTGRE na Universidade da Califórnia, em Berkeley. Este projeto foi patrocinado por vários órgãos governamentais americanos como a *Defense Advanced Research Projects Agency* (DARPA), Army Research Office (ARO), National Science Foundation (NSF), entre outros (STONEBRAKER e ROWE, 1986).

De acordo com Michael Stonebraker e Lawrence A. Rowe (1986), o projeto POSTGRE foi um recomeço para construir um sistema de banco de dados que atendesse suas expectativas concernentes ao modelo de dados relacional. Seu projeto anterior, chamado de INGRES, já havia sido modificado várias vezes e por conta de seu *design* de criação, não suportava o modelo E/R.

Ao ser adicionado um interpretador da linguagem SQL ao projeto POSTGRE, este passou a se chamar PostgreSQL e melhorias no projeto têm sido realizadas desde então.

Segundo The Postgre Global Development Group (2020), o PostgreSQL trabalha com parâmetros dos bancos de dados modernos, de acordo com as seguintes características:

- Consultas: possibilita criar instruções SELECT dentro de outras instruções SELECT tornando várias instruções que seriam escritas em sequência e consequentemente demorariam mais para alcançar o resultado em uma única instrução completa que economiza tempo;
- Chaves estrangeiras: cria a possibilidade de relacionamento entre tabelas distintas;
- Gatilhos (*triggers*): os comandos podem ser executados automaticamente de acordo com a programação estabelecida;
- Exibições atualizáveis: atualiza os campos das tabelas que são visualizadas de acordo com os novos comandos estabelecidos;
- Integridade Referencial: mantém o banco de dados logicamente estruturado garantindo que uma chave estrangeira em uma tabela destino é uma chave primária de algum registro na tabela origem;

- Integridade Transacional: garante propagar a atualização dos campos e a exclusão dos registros selecionados;
- Controle de concorrência multiversão: impede que uma transação obtenha dados inconsistentes, que poderiam ter sido gerados por atualizações nas mesmas linhas de dados, disponibilizando a cada operação uma versão do banco de dados consistente e mantendo o padrão ACID.

2.7 FIREBIRD

De acordo com Manual Referencial do Firebird (2020), é um sistema gerenciador de banco de dados de código aberto, desenvolvido e mantido pelo trabalho voluntário de desenvolvedores e pela fundação FireBirdSQL, empresa criadora do *kernel* do *software*.

O Firebird está disponível nas principais plataformas como Windows, Linux e MacOS, oferecendo suporte para a comunicação com várias linguagens de programação como: C, C++, Java, Delphi, Python, Ruby, entre outros.

De acordo com Carlos H. Cantu (2010), o Firebird é um sistema gerenciador de banco de dados completo e robusto em suas funções, podendo gerenciar bancos de dados com diferentes tamanhos mantendo seu padrão de qualidade. Este oferece os seguintes recursos:

- Suporte a *Store Procedures*: cria a possibilidade de uma biblioteca de comandos em SQL para a utilização junto ao banco de dados, esta armazena tarefas repetitivas aceitando parâmetros de entrada para que a tarefa seja completada de acordo com a necessidade que este comando atende;
- Suporte a *Triggers*: disponibiliza um recurso para programar a execução automática de funções em uma tabela, tais como: inserções, exclusões, modificações, entre outras;
- Integridade Referencial: mantém o banco de dados logicamente estruturado garantindo que uma chave estrangeira em uma tabela destino é uma chave primária de algum registro na tabela origem;
- Versatilidade: disponibiliza uma versão embarcada do sistema para aplicações de demonstração ou *standalone*;

- Processamento: consome poucos recursos do computador facilitando a velocidade de execução de outros *softwares* que precisem destes recursos para sua execução;
- Configuração: proporciona uma instalação e configuração simples com recursos pré-definidos, logo o usuário não necessita ter muito conhecimento da ferramenta para poder utiliza-la;
- TraceAPI: recurso que possibilita ao usuário obter informações da atual situação do servidor;
- Transações: compatíveis com o padrão ACID.

2.8 FERRAMENTAS DE ANÁLISE DE DESEMPENHO DE BANCO DE DADOS

De acordo com Seng (2015) os benchmarks são divididos em duas categorias: *benchmarks* sintéticos e benchmarks empíricos.

Os *benchmarks* sintéticos caracterizam-se por utilizarem-se de dados, operações e bases de dados artificias para fazer uma simulação do problema afim de determinar o parâmetro de saída. Já os *benchmarks* empíricos fazem a análise com informações e operações reais em um sistema criado e em funcionamento (SENG, 2015).

Os *benchmarks* empíricos são de fato ideais para a medida de desempenho em um sistema, mas o custo para a implementação deste sistema é muito alto pois como estes usam bases de dados e informações reais, implementa-las nas bases de dados dos *benchmarks* torna-se um trabalho inviável (SENG, 2015).

Devido a este fator, os *benchmarks* sintéticos são mais desenvolvidos que os empíricos, sendo os principais apresentados nas seções subsequentes.

2.8.1 *Benchmarks* TPC

De acordo com Jérôme Darmont (2017), tratando-se de *Benchmarkings* em um SGDBR, existe um grupo chamado *Transaction Processing Performance Council* (TPC) que é responsável por criar *Benchmarkings* padrões, verificar a correta aplicação destes pelos usuários e periodicamente publicar resultados de testes de desempenho entre bases de dados relacionais.

Esta organização sem fins lucrativos é responsável pela criação de diversos algoritmos que propiciaram a análise de desempenho dos bancos de dados ao longo do tempo.

2.8.2 *Benchmark* TPC-C

De acordo com Bert Scalzo (2018) o software de *benchmark* TCP-C é o mais reconhecido de todos os *benchmarks* da categoria TPC, também é o mais oferecido para a realização de métricas em ferramentas de análise de desempenho de banco de dados relacionais. Tem sido utilizado desde 1992 especificamente dedicado a medir o desempenho de aplicações com processamento transacional *online*.

A base de dados do TPC-C é composta por nove tabelas variando desde pequenos até grandes volumes de dados. Esta metodologia proporciona o TPC-C a medir a quantidade de transações por minuto.

2.8.3 *Benchmark* Wiscosin

O *benchmark* Wisconsin é um dos mais antigos algoritmos de análise de desempenho. Foi criado em meados de 1981 quando os primeiros gerenciadores estavam entrando no mercado.

Devido ao interesse que havia em construir *benchmarks* que fornecessem o tempo de resposta das operações de transação em um banco de dados, o Wisconsin foi desenvolvido visando dois principais objetivos: que as *queries* dentro do *benchmark* deveriam testar o desempenho de cada tabela separadamente e que as relações entre tabelas deveriam ser de fácil entendimento para que adições de novas *queries* pudessem ser feitas mais facilmente (DEWITT, 1991).

A base de dados do *benchmark* Wisconsin trabalhava com três tabelas, uma delas com mil linhas e as outras com dez mil linhas de informação. Nessa época o poder de processamento dos computadores era fraco e os espaços em disco eram limitados, por esse motivo, o Wisconsin tinha que atuar com uma base de dados pequena (DEWITT, 1991).

2.8.4 *Benchmark* AS³AP

O ANSI SQL Standard and Portable (AS³AP) é uma ferramenta para análise de desempenho de banco de dados padrão. Apesar de ser antiga, criada em 1984, é considerada confiável e em muitos aspectos considerada uma extensão do *Benchmarking* Wisconsin (SCALZO, 2018).

A maneira com que o AS³AP faz a análise de desempenho é por meio de 5 tabelas em sua base de dados, uma tabela possui uma única coluna e linha e as outras quatro possuem de dez mil a um bilhão conforme as iterações que são feitas a cada ciclo de testes. Essas informações quando colocadas no banco de dados, criam uma base de dados variando desde 4 megabytes a 400 gigabytes de tamanho (SCALZO, 2018).

2.8.5 *Benchmark* EDB

De acordo com Gray (1993), o EDB (*Engineering Database Benchmark*) é um benchmark desenvolvido para medir o desempenho especificamente de aplicações de engenharia que contenham bases de dados.

Este *benchmark* difere-se muito do Wisconsin por apresentar medidas mais detalhadas e focar-se em *queries* mais simples na hora dos testes de entrada e saída de dados (GRAY, 1993).

O EDB não somente pode ser aplicado a arquiteturas de bancos de dados relacionais como também a outros paradigmas como: orientação a objetos, network, hierárquicos e árvores binárias.

A razão pela qual o EDB pode ser aplicado a várias arquiteturas é que este é independente da base de dados proveniente do banco de dados. Essa característica faz com que o *benchmark* seja mais consistente em suas medições (GRAY, 1993).

2.9 PROCESSO PARA A CONSTRUÇÂO DE UM *BENCHMARK*

Nesta seção é apresentada uma metodologia para a construção de um *benchmark* sintético.

2.9.1 Critérios Gerais de um *Benchmark*

De acordo com Gray (1993), quatro critérios especificam um *benchmark* sintético, os quais são:

- Relevância: o *benchmark* deve trabalhar com aspectos de desempenho que atraiam o maior número de usuários em potencial;
- Portabilidade: o *benchmark* deve poder ser aplicado a diferentes sistemas gerenciadores de bancos de dados;
- Simplicidade: o *benchmark* deve ser viável e não deve exigir muitos recursos;
- Escalabilidade: o *benchmark* deve se adaptar a pequenas e grandes arquiteturas de computadores.

O *benchmark* é constituído basicamente de dois elementos: o modelo da base de dados, que é implementado nos sistemas gerenciadores que serão submetidos à análise de desempenho, e o conjunto de operações que executam a entrada e a saída de informações no banco de dados (DARMONT, 2017).

2.9.2 Caracterização da Carga de Trabalho

De acordo com Paula R. Strawser (1984) a carga de trabalho em um sistema de computador é definida como o conjunto de todas as entradas que o sistema recebe do ambiente em que atua. Também pode ser definida como o número total de requerimentos feitos pelos usuários ao sistema.

Como o *benchmark* é sintético, sua carga de trabalho é analisada por meio de um modelo artificial, este envolve uma base de dados e um conjunto de aplicações experimentais.

De acordo com Seng (2015) a carga de trabalho é o ponto principal em um *benchmark*, dois tipos de variáveis são fundamentais para a construção de um modelo artificial que é a base para os resultados em um *benchmark*: fatores experimentais e métricas.

2.9.3 Fatores Experimentais

Os fatores experimentais são variáveis que afetam o desempenho do sistema. São também chamadas de predicados ou variáveis independentes, termos comumente utilizados tratando-se de análises comparativas. Exemplos dessas variáveis são: tamanho do banco de dados, complexidade da *query,* número de usuários do banco, entre outros (SENG, 2015).

2.9.4 Métricas

Métricas são variáveis de saída, são também chamadas de variáveis dependentes, pois serão afetadas conforme a atuação das variáveis independentes, estas podem ser: tempo de resposta, *throughput (*quantidade de transações produzidas ao longo do tempo durante um teste), número de comandos de entrada e saída, memória de disco utilizada, entre outras (DARMONT, 2017).

2.9.5 Modelo da Base de Dados

De acordo com Paula R. Strawser (1984), o modelo da base de dados não pode estar interligado a uma aplicação real para que possa ser reproduzido em diferentes situações sem a necessidade de alteração do seu conteúdo.

Existem características que refletem as dimensões físicas das bases de dados como: número de relações, largura da tupla e a cardinalidade de cada relação. Também há características que refletem o conteúdo da base de dados, estes são os tipos de dados e a distribuição dos valores dos atributos (STRAWSER, 1984).

Para que a análise de desempenho ocorra de uma maneira coesa, é preciso limitar certas características. Strawser (1984) sugere fixar o número de relações na base de dados.

Aspectos como largura das tuplas, número de atributos e cardinalidade das relações são determinados de acordo com as características dos objetos de análise.

2.9.6 Aplicações de Manipulação da Base de Dados

Mais uma vez, Paula R. Strawser (1984) enfatiza a questão da independência de uma aplicação, neste caso é sugerido que aplicações que lidem com o banco de dados sejam independentes de qualquer aplicação real.

As aplicações que interagem com a base de dados são modeladas como a construção de uma sequência de instruções que são executadas objetivando a manipulação da base de dados.

2.9.7 Quantidade de Medições

De acordo com Paula R. Strawser (1984), a confiabilidade na análise de desempenho está intimamente ligada ao número de medições que o *benchmark* produz. O *benchmark* deve cobrir um grande número de situações afim de garantir a consistência dos parâmetros de saída. Uma única situação com uma única medida não pode ser confiável.

2.9.8 Técnicas de Avaliação

Existem duas técnicas possíveis para avaliar o desempenho de sistemas, estas são: modelagem e medições.

De acordo com Ferrari (1978), a primeira cria um modelo matemático do sistema, representando todas as suas entradas e saídas, com este modelo é possível prever o desempenho do mesmo ainda em fase de projeto. Existem limitações quanto ao uso desta técnica devido as diversas simplificações que o modelo sofre baixando o nível de precisão nos resultados. A vantagem deste método é o pouco tempo que este pode ser implementado, a pouca instrumentação necessária para cria-lo e a possibilidade de criar mais de um modelo afim de comparar seus resultados.

As medições consistem em monitorar um sistema afim de obter dados de seu desempenho. Neste método é necessário que já exista o sistema, diferentemente do método de modelagem. Sua grande vantagem é que ele proporciona resultados precisos por que foca exclusivamente na carga de trabalho do sistema em vez da determinação de um modelo matemático. Sua desvantagem é o requerimento de

equipamentos e instrumentação podendo acarretar custos dependendo do sistema ao qual este método analisa.

2.9.9 Planejamento

A metodologia utilizada foi adaptada de Raj Jain (1991) a partir dos conceitos descritos nos tópicos anteriores:

- Escolha dos sistemas alvo do *benchmark*: consiste em escolher com clareza os sistemas que serão submetidos à análise comparativa, trata-se de um ponto crucial da metodologia, pois a escolha incoerente leva a resultados inconsistentes e a análise de desempenho passa a não apresentar sentido lógico;
- Analisar parâmetros de carga de trabalho: consiste em diversas análises sobre fatores experimentais, métricas, base de dados a ser utilizada, operações no banco de dados, plataformas, entre outras;
- Escolha do parâmetro de medição: objetiva a escolha da métrica que será reportada pelo *benchmark* de acordo com as medições vistas nos parágrafos anteriores;
- Escolha da técnica de avaliação: consiste em analisar qual das técnicas descritas anteriormente proporcionará resultados satisfatórios levando em consideração os sistemas que serão analisados, a carga de trabalho dos sistemas e os parâmetros de medição;
- Avaliação de parâmetros prejudiciais: visa relatar os possíveis fatores que podem afetar o desempenho do sistema de forma que este não apresente resultados confiáveis, tomando medidas para minimizar ou erradicar o problema;
- Execução da análise: consiste na aplicação do algoritmo de *benchmark* para os sistemas;
- Análise e interpretação dos resultados: avaliar os resultados obtidos das medições e simulações realizadas afim de estabelecer conclusões acerca dos sistemas.

A aplicação de uma metodologia estabelece um padrão para a aquisição de resultados verdadeiros. O plano para o desenvolvimento de um *benchmark* é colocar os sistemas em uma condição de igualdade inicial, retirando todos os possíveis fatores que poderiam beneficiar qualquer ferramenta.

A seguir encontra-se o diagrama do processo de desenvolvimento do *benchmark* sintético mostrando os processos de avaliação e tomadas de decisão discutidos anteriormente:

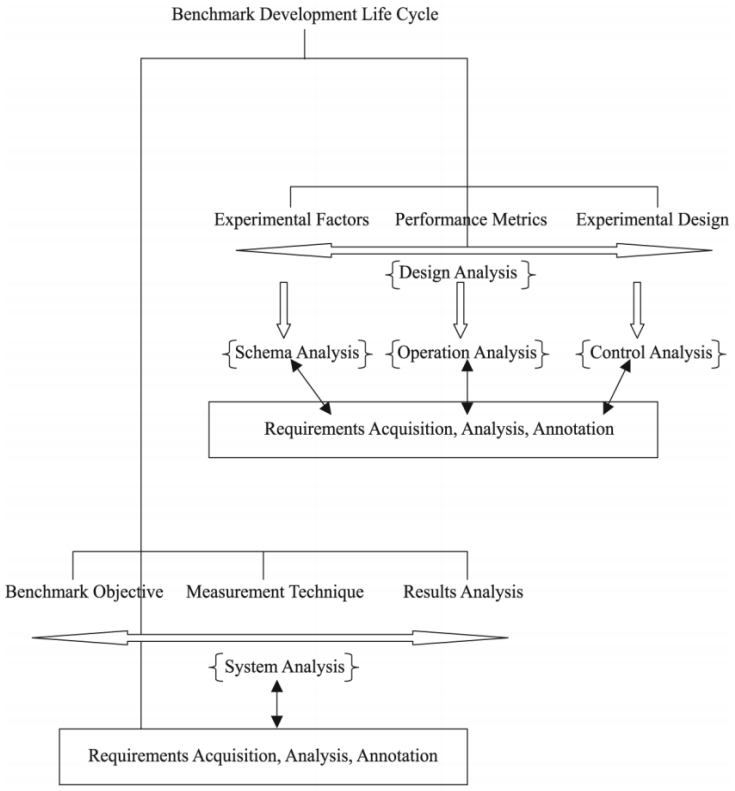


Figura 11 – Ciclo de Desenvolvimento do *Benchmark*

Fonte: (SENG, 2015)

A partir da figura 11 é possível entender que existem análises que devem ser feitas antes da criação de um algoritmo para a aquisição de resultados. Este processo é dividido em duas etapas, à direita superior do diagrama é mostrado as análises da base de dados, das métricas e experimento enquanto que à esquerda inferior mostra a determinação do objetivo do benchmark, da técnica de medição e a análise dos resultados. Quando por algum motivo ocorre um erro na determinação desses parâmetros, ocorre o aumento do tempo para a determinação dos resultados.

3 METODOLOGIA

Este capítulo trata a respeito do desenvolvimento dos algoritmos que atuam no banco de dados, desde a criação de tabelas até a obtenção dos parâmetros de saída, proporcionando o entendimento do sistema em sua forma técnica.

3.1 Geração de Números Randômicos em Java

Alguns sistemas descritos nas seções a seguir utilizam-se de números randômicos da linguagem Java para executar suas funções. O processo de geração de números randômicos é fornecido por meio de um valor colocado na declaração de uma variável randômica na linguagem Java conforme a figura 12:

```
public static Random rand = new Random(364517351351L);
```

Figura 12 - Geração de números randômicos Java

Fonte: Autoria Própria

O número declarado para a variável “rand” é denominado de *seed*, este serve como base para a geração de um número randômico. Logo o algoritmo tendo este valor base, sempre selecionará um conjunto de números cada vez que o *software* for executado. Caso queira-se mudar a sequência de números que será gerada pelo algoritmo, é necessário declarar outro valor para *seed*.

3.2 Incremento Realizado por *Software*

O banco de dados relacional possui um recurso descrito como *AUTO INCREMENT* para chaves primárias. Esta cláusula possibilita ao contador de uma chave incrementar automaticamente a cada inserção de dados em uma tabela, contudo este recurso foi substituído pelo incremento realizado diretamente pelo algoritmo que faz as operações nos bancos de dados pelos seguintes motivos:

- Sintaxe: apesar dos bancos de dados relacionais possuírem a padronização da linguagem SQL, cada um possui suas particularidades na hora da declaração de uma chave primária com incremento automático, por exemplo: o MySQL utiliza-se de uma cláusula *AUTO_INCREMENT* para fazer a operação enquanto que o PostgreSQL usa o parâmetro *SERIAL*;
- Generators e Triggers: são trechos de código utilizados por um banco de dados na hora de um incremento automático. Alguns bancos geram automaticamente estes componentes e os mantém alocados internamente e fora do contato com o usuário. Contudo, como nem todos os bancos de dados possuem este recurso, é necessário gerá-los por meio de comandos que definam qual a tabela e o atributo que será realizado o incremento. Além disso, o sistema deve excluí-los antes da exclusão de uma tabela, fazendo com que a complexidade da implementação de uma solução para este caso seja inviável;
- Tratativa de Exclusão: no caso da exclusão de um valor em uma tabela, o banco de dados possui abordagens para lidar com os identificadores das informações que permaneceram inalteradas. O sistema pode optar por não alterar o valor da chave primária, fazendo com que a tabela possua lacunas, quebra da sequência de numeração das chaves primárias, ou o algoritmo pode realinhar esta sequência mudando o valor dos identificadores.

As características discutidas neste tópico fizeram com que a opção de auto incremento para o projeto fosse algo inviável, pois deixar que cada sistema faça seu próprio gerenciamento incremental poderia significar riscos na etapa de manipulação da base de dados, logo optou-se por fazer incrementos pelo algoritmo na hora da inserção dos valores.

3.3 Diagramas de UML do Projeto

A partir das diretivas apresentadas na capitulo 2 a respeito das ferramentas pesquisadas bem como o processo de desenvolvimento do *benchmark,* foi desenvolvido o diagrama de perfil do projeto utilizando a Linguagem de Modelagem Unificada*, do* inglês *Unified Modeling Language (UML).* Este pode ser visualizado conforme a figura 13:

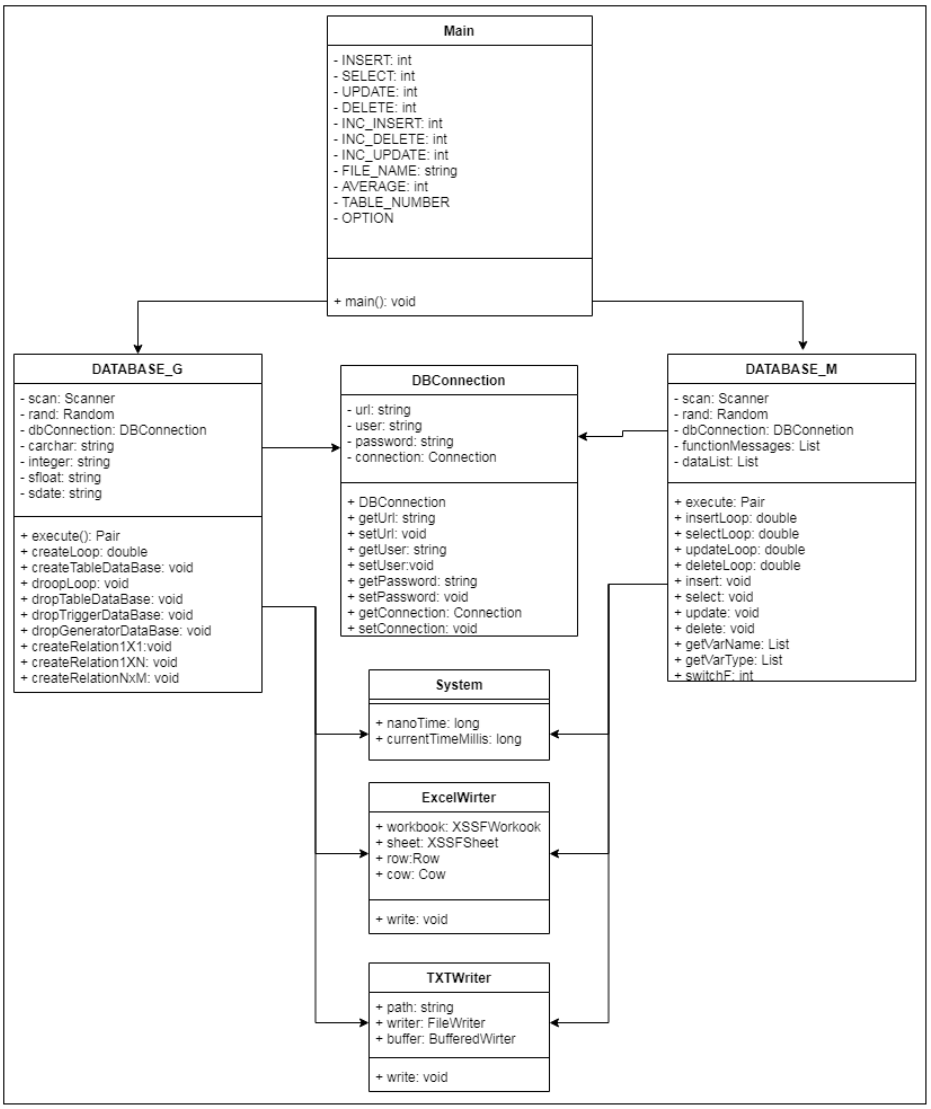


Figura 13 - Diagrama UML de Perfil do Projeto

Fonte – Autoria Própria

A partir da figura 13, é possível observar que o projeto é composto por sete classes, as interações entre elas são realizadas por meio da classe Main, nessa classe são definidas variáveis que controlam a quantidade máxima de cada operação, a quantidade de incrementos que serão realizados a cada iteração do *software*, o número de vezes que o algoritmo fará o procedimento para a obtenção da média e o modo de operação que o programa é executado.

Exemplificando a execução do *software*, para as sublinguagens SQL DML e DQL que formam as operações CRUD, em uma inserção máxima de 500 mil

informações com incrementos de 20 mil, o sistema executa 25 vezes ciclos de 20 mil operações, logo o sistema faz 20 mil operações de inserção, seleção, modificação e exclusão, obtêm o resultado deste lote de operações e então o incrementa em mais 20 mil operações, executando o processo novamente até atingir o valor final. Quando a variável que estipula o valor da média obtida a cada iteração é determinada maior do que 1, o algoritmo executa um lote de operações, mas não o incrementa logo em seguida para ir para o próximo, em vez disso, o algoritmo executa o mesmo lote de operações quantas vezes determinar o número da variável da média, obtendo tempos de operações que são divididos pela quantidade de vezes que o algoritmo executou o lote de operações.

A figura 14 mostra o diagrama UML sequencial da execução do algoritmo com os valores máximos de operação e incremento de lotes descritos previamente:

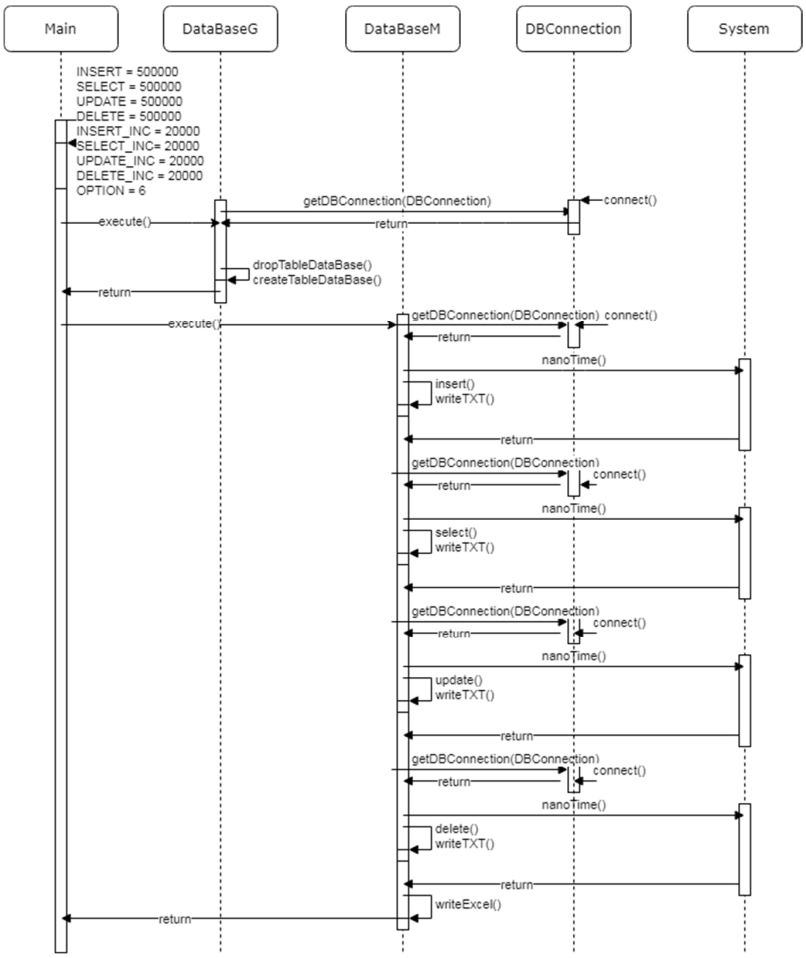


Figura 14 - Diagrama Sequencial UML para Testes com as Operações CRUD

Fonte: Autoria Própria

O diagrama de sequência da figura 14 mostra o processo para obtenção dos resultados dos desempenhos de cada banco. O algoritmo começa determinando as variáveis descritas anteriormente bem como o modo de operação, este trata a respeito da quantidade de bancos que serão analisados automaticamente.

Existem sete opções diferentes que podem ser determinadas para o modo de operação: executar isoladamente o MySQL, o PostgreSQL ou o Firebird, executar dois bancos de dados ou todos eles. Esta opção foi desenvolvida para que a análise pudesse ser focada em um banco de dados que apresentasse problemas decorrentes da quantidade insuficiente de iterações sobre as operações provocando uma média ruim que seria vista nos gráficos. Logo esta opção permite uma análise isolada de cada banco de dados com grandes quantidades de iterações.

Estabelecido o modo de operação, o algoritmo estabelece uma conexão com cada banco de dados e cria uma base de dados por meio da classe DataBaseG, em seguida o algoritmo acessa a classe DataBaseM para realizar os ciclos de operações, estabelecendo e fechando conexões com os bancos de dados conforme o início e o termino de cada lote de operações.

Para se estabelecer uma conexão com os bancos de dados, foi criada uma classe chamada DBConnection, esta recebe parâmetros de acesso aos bancos de dados como o caminho de acesso aos bancos no computador, o nome do usuário cadastrado no banco e a senha. Com essas informações a classe estabelece uma conexão para que as outras classes possam interagir com os bancos de dados.

O *software* executa as operações CRUD, obtendo o tempo por meio de uma *thread* de contagem de tempo e escrevendo os resultados em arquivos texto e em planilhas Excel para a obtenção dos gráficos da análise.

Os testes envolvendo a criação e a remoção de tabelas não envolvem a classe DataBaseM, somente a classe DataBaseG, que contém essas funções. Por meio da classe Main é selecionado a quantidade de tabelas a serem criadas e o algoritmo segue o mesmo processo de análise das sublinguagens DML e DQL, por meio dos parâmetros estabelecidos como a quantidade de vezes que o algoritmo fará o processo e os bancos de dados a serem analisados, o *software* estabelece o tempo das operações para as quantidades selecionadas.

A seguir encontra-se o diagrama UML sequencial que mostra o processo de execução do algoritmo para as sublinguagens SQL DDL:

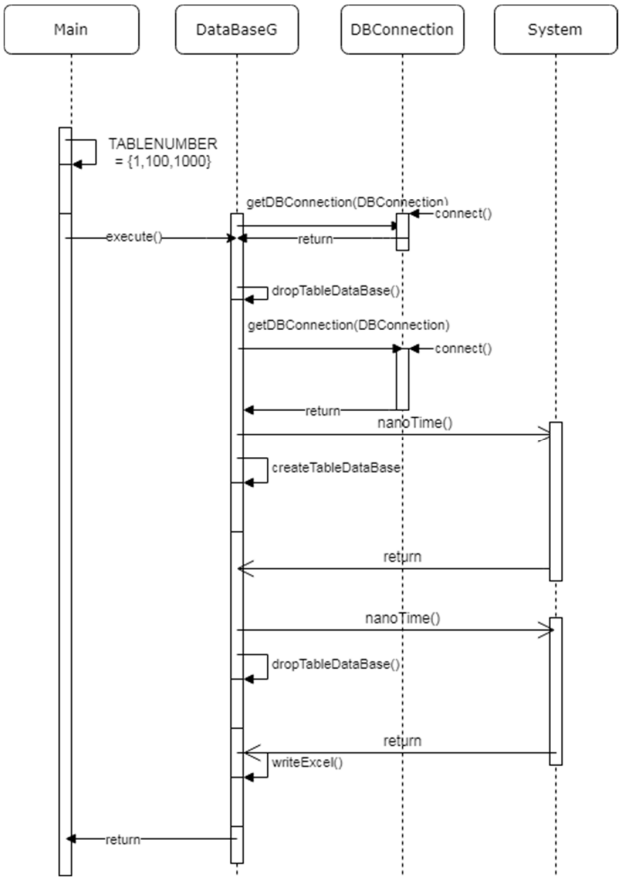


Figura 15 - Diagrama Sequencial UML para Testes com as Operações DDL

Fonte: Autoria Própria

A figura 15 mostra o diagrama UML sequencial para a análise das operações de criação e remoção de tabelas, pelo diagrama é possível notar que o processo é semelhante ao das operações CRUD, contudo é necessário antes de realizar a análise destas operações, limpar toda a base de dados, visto que pode conter tabelas e informações de testes envolvendo as operações CRUD ou até mesmo testes com operações da linguagem DDL.

As seções subsequentes oferecem informações detalhadas de cada algoritmo que compõe o sistema.

3.4 Sistema de Geração de Base de Dados Automático

Como o *benchmark* sintético é produzido por meio de uma base de dados fictícia, foi desenvolvido para a aplicação do experimento um sistema que possibilita

gerar diversas bases de dados com diferentes números de tabelas e variáveis. Esse algoritmo foi utilizado tanto para os testes com a linguagem SQL do tipo DML e DQL quanto para testes com a linguagem SQL do tipo DDL.

3.4.1 Remoção de Tabelas em uma Base de Dados

A remoção das tabelas em um banco de dados, é um processo que ocorre verificando cada uma que está contida na base de dados e excluindo uma a uma até que a base de dados fique vazia.

O algoritmo de exclusão tem a função de eliminar as tabelas para remover de uma única vez os dados fictícios que foram alocados em outras simulações caso a operação de exclusão de dados não tenha sido atribuída com o mesmo valor da operação de inserção.

3.4.2 Criação de Tabelas

A criação de tabelas, compreende-se em criar somente as tabelas sem nenhum tipo de relacionamento entre elas. O algoritmo cria as tabelas de acordo com uma quantidade selecionada antes da execução do *software*.

A figura 16 mostra o exemplo da tabela criada para os testes com a linguagem SQL do tipo DML e DQL:

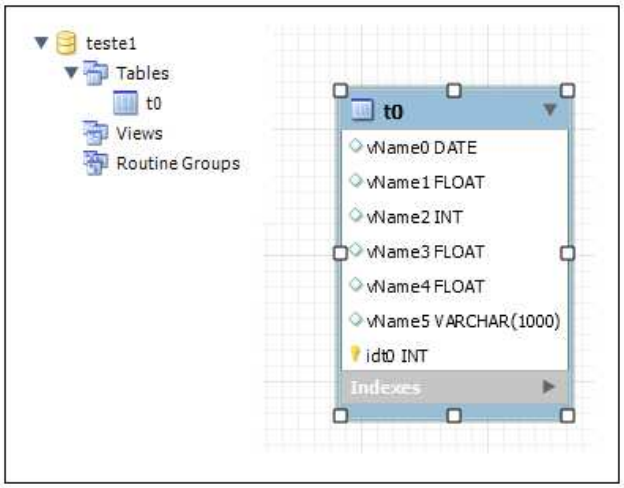


Figura 16 – Tabela Criada para os Testes com as Operações CRUD

Fonte: Autoria Própria

De acordo com a figura 16 é possível observar que o sistema cria os nomes das tabelas com a letra T acrescida de um número que varia conforme as tabelas anteriores são criadas, naturalmente é possível criar quantas tabelas forem requeridas para a realização dos testes.

As chaves primárias são denominadas pela palavra id acrescido do nome da tabela. Todas as chaves são caracterizadas na sua criação como inteiras e não possuem parâmetros de auto incremento em nenhum banco, toda a configuração de incremento automático foi feito no próprio *software* pelos motivos descritos na seção 3.2 do trabalho.

Os atributos criados são denominados como “vName” acrescido de um número que representa a ordem da sua criação em um conjunto de variáveis, logo para um conjunto de 3 variáveis, os nomes destas seriam respetivamente: “vName0”, “vName1” e ““vName2”.

A tabela 3 mostra o padrão para os nomes de atributos, chaves primárias e tabelas em uma base de dados fictícia criada pelo algoritmo.

Tabela 3 - Nomes dos elementos da base de dados fictícia

Elementos da Base de Dados	Sigla	Acréscimo
Tabelas	T	Número da Tabela
Atributos	vName	Número do Atributo
Chaves Primárias	Id	Nome da Tabela

Fonte: Autoria Própria

A cada criação de uma tabela também são criados seus atributos, os mesmos são obtidos por meio de números randômicos que selecionam a quantidade de variáveis e o seu tipo, podendo ser: inteiros, reais, datas e *strings*.

A quantidade de variáveis é selecionada em um intervalo de 1 a 10 unidades, isso significa que o algoritmo para cada tabela da base de dados selecionará a cardinalidade dentro deste intervalo.

O algoritmo também possui a criação de chaves estrangeiras para interligar as tabelas por meio do modelo E/R, contudo, como elas não foram utilizadas, somente podem ser visualizadas na seção de Apêndice do trabalho.

A figura a seguir mostra a função que cria uma tabela na base de dados assim como seus atributos:

```
public static void createTableDataBase(Connection connection, int tableNumber, Pair<Integer, List<Integer>> randsCreation) {

    //--define how many variables will be in a table
    String query = "CREATE TABLE T" + tableNumber + " (";
    int randVariables = randsCreation.getKey();
    //--Choose variables according to the list of variables
    for (int j = 0; j < randVariables; j++) {
        int randType = randsCreation.getValue().get(j);
        switch (randType) {
            case 0:
                query = query + "vName" + j + " " + varchar + "(1000)" + ",";
                break;
            case 1:
                query = query + "vName" + j + " " + integer + ",";
                break;
            case 2:
                query = query + "vName" + j + " " + sfloat + ",";
                break;
            case 3:
                query = query + "vName" + j + " " + sdate + ",";
        }
    }

    StringBuilder sbuilder = new StringBuilder(query);
    sbuilder.setCharAt(query.length() - 1, ')');
    query = sbuilder.toString();

    try {
        Statement statement = connection.createStatement();
        statement.executeUpdate(query);
    } catch (SQLException e) {
        throw new IllegalStateException("DataBaseAutoGenerator1.createTableDataBase: Error", e);
    }

}
```

Figura 17 - Função de Criação de Tabelas e Atributos de uma Base de Dados

Fonte: Autoria Própria

A partir da figura 17, é possível observar que uma vez selecionada a quantidade de variáveis, o *software* por meio de um processo iterativo seleciona o tipo de cada varável de acordo com as quatro categorias: inteiros, reais, datas e *strings*. Essas categorias foram selecionadas por meio de uma pesquisa entre os bancos de dados para que fossem escolhidos os principais tipos de atributos que são selecionados na hora da criação das tabelas em uma base de dados.

O processo de criação de tabelas assim como de seus atributos garante que a mesma base de dados seja criada em todos os bancos mesmo utilizando-se de números randômicos para a geração de atributos, pois como descrito na seção 3.1, o algoritmo a partir de um valor base seleciona e executa a sequência de números a cada execução do algoritmo de geração da base de dados, logo a mesma base de dados é gerada em todos os bancos de dados conforme a sequência que foi definida.

A partir do algoritmo de criação de tabelas, foram criadas bases de dados com diferentes tamanhos para os testes envolvendo a linguagem DDL, funções de criação e exclusão de tabelas em um banco de dados. Essas tabelas eram verificadas por meio de um *Workbench*, *software* que permite a visualização de bases de dados, o

exemplo da criação da cardinalidade máxima dos testes, mil tabelas, é mostrado por meio da figura 18:

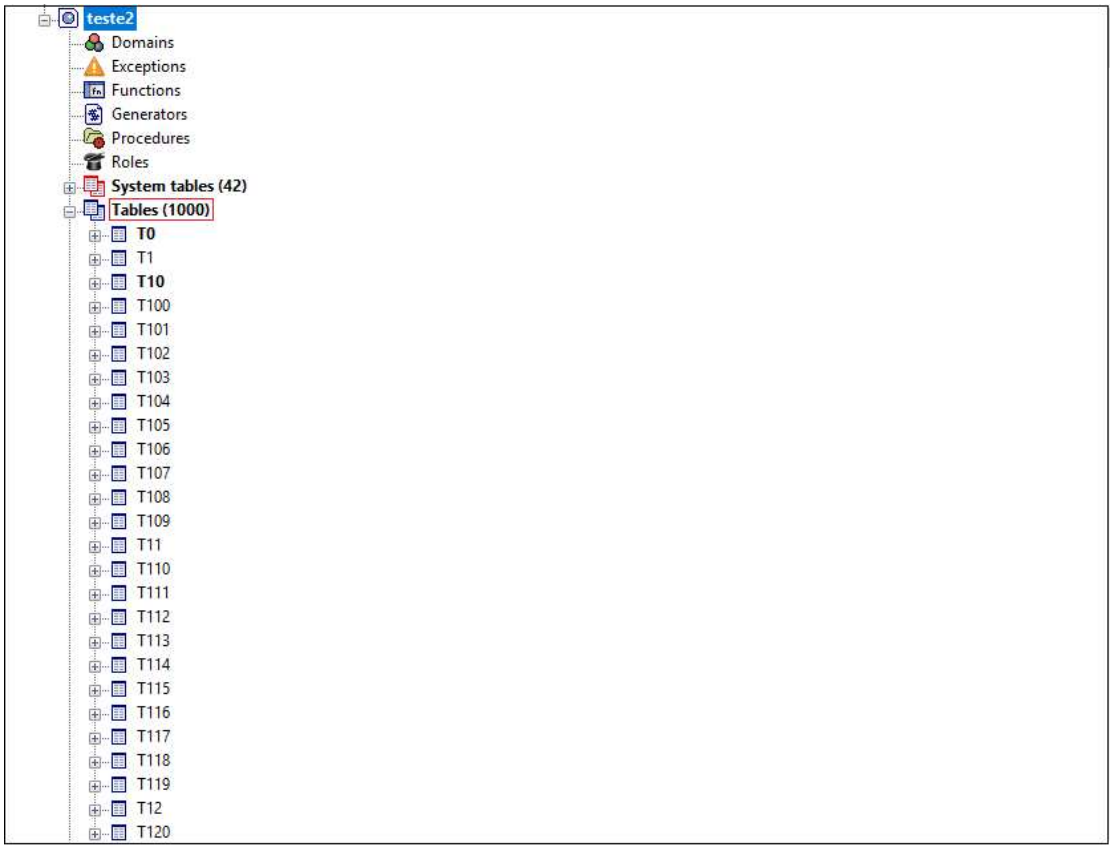


Figura 18 - Base de Dados para Testes com a Linguagem SQL do tipo DDL

Fonte: Autoria Própria

A figura 18 mostra as tabelas criadas para os testes de criação e remoção de tabelas, a mesma figura foi obtida por meio do Workbench FlameRobin, este é um *software* específico para a visualização da base de dados do sistema Firebird.

3.5 Sistema Manipulador de Base de Dados

O sistema manipulador de base de dados é um algoritmo desenvolvido para fazer as operações de inserção, seleção, modificação e exclusão de dados. Este é executado após o sistema de criação de bases de dados descritos nos tópicos anteriores.

O algoritmo executa sequencialmente as operações de acordo com quantidades que são digitadas antes da execução do sistema. Conforme descrito na seção 3.3, o algoritmo realiza iterações de valores das operações CRUD até valores máximos que são definidos antes da análise.

Os próximos tópicos descrevem o funcionamento do *software* assim como a lógica implementada para cada função.

3.5.1 Algoritmo de inserção

A inserção de informações na base de dados é um processo automático de população de tabelas. Inserções em massa são realizadas em todas as tabelas do banco de dados seguindo uma abordagem que objetiva preencher com dados fictícios todas as tabelas da base de dados em um sistema.

As informações inseridas são visualizadas por meio de uma interface gráfica que foi desenvolvida para o projeto conforme mostra a figura 19:

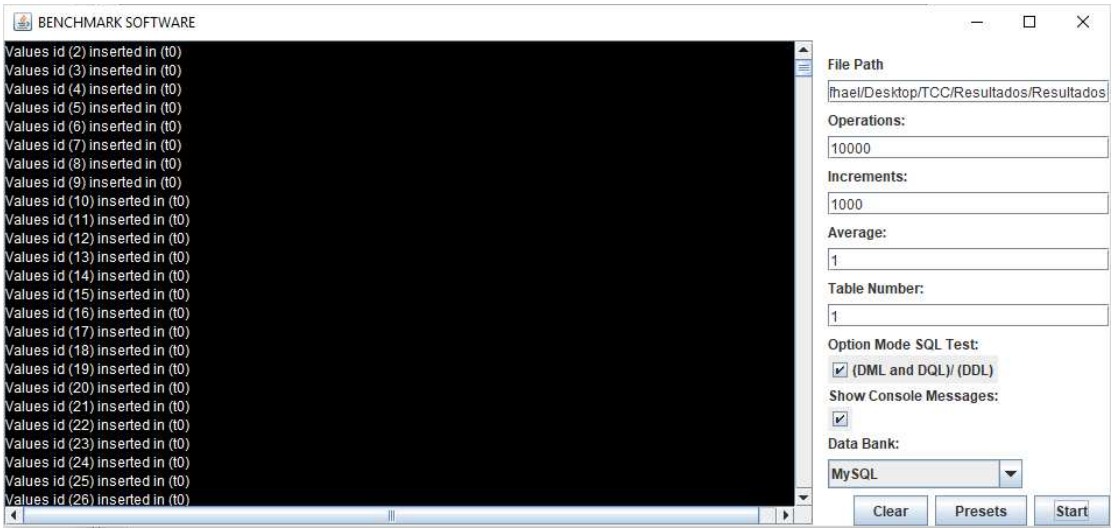


Figura 19 - Visualização de Dados Inseridos

Fonte – Autoria Própria

O algoritmo preenche todas as tabelas com a mesma quantidade de dados, este é explicado detalhadamente de acordo com a figura 20:

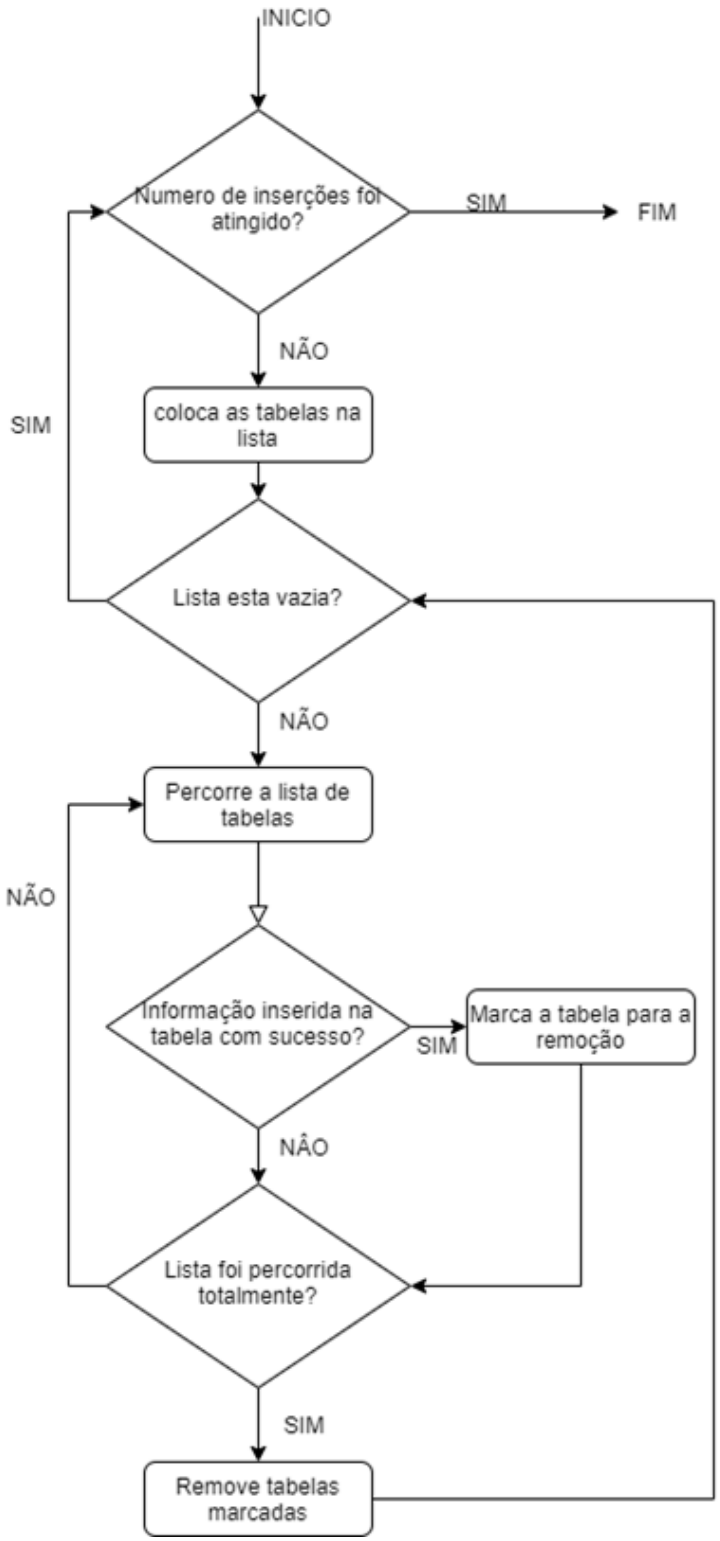


Figura 20 - Fluxograma do Algoritmo de Inserção de Valores

Fonte: Autoria Própria

A figura 20 refere-se ao processo de inserção automática na base de dados previamente criada, o algoritmo obtém todas as tabelas contidas em um banco de dados por meio de função de obtenção de informações do banco contida na classe DBConnection.

O algoritmo aloca todos os dados das tabelas em uma lista e passa a percorre-la tentando fazer uma inserção. Por meio da resposta positiva fornecida pelo banco de dados o algoritmo marca um determinado elemento da lista para remove-lo após percorre-la completamente seguindo assim por diante até o término da operação em todas as tabelas, quando o algoritmo volta a verificar se o número máximo de inserções foi atingido, retornando ou não os valores de todas as tabelas para a lista.

3.5.2 Inserção de Dados

A inserção de uma informação em uma tabela considera o tipo de atributo contido na mesma realizando a operação catalogando cada um deles em um valor pré-determinado no código conforme a figura 21:

```
switch (index) {
    case 1:
        pStatement.setInt(counter, 1111111111);
        break;
    case 2:
        pStatement.setString(counter, "BENCHMARK TESTING");
        break;
    case 3:
        pStatement.setFloat(counter, 222222222);
        break;
    case 4:
        pStatement.setDate(counter, java.sql.Date.valueOf("2020-02-11"));
}
```

Figura 21 - Atribuição de Valores nos Atributos das Tabelas

Fonte: Autoria Própria

A figura 21 mostra a atribuição de valores aos atributos de uma tabela, a função de inserção cataloga e atribui o valor de acordo com o tipo de variável que está lendo no momento.

Os valores de cada atributo são simbólicos, ou seja, não possuem nenhum significado em uma base de dados e servem como identificadores que a operação foi realizada com êxito.

3.5.3 Seleção de Atributos em uma Tabela

A seleção dos dados de todos os atributos de uma tabela é realizada por uma função que envia um comando SQL ao banco de dados conectado. Trata-se de uma função que não exige muitos recursos da linguagem Java uma vez que o próprio banco já disponibiliza um comando para esta seleção por meio da cláusula * que significa a seleção completa.

Conforme o comando é estabelecido, é feita a requisição nos bancos de dados e o valor, caso encontrado, é retornado em formato de uma lista de objetos Java, sendo mostrada no console da interface gráfica do projeto. O exemplo da visualização das informações dessa função é mostrado na figura a seguir:

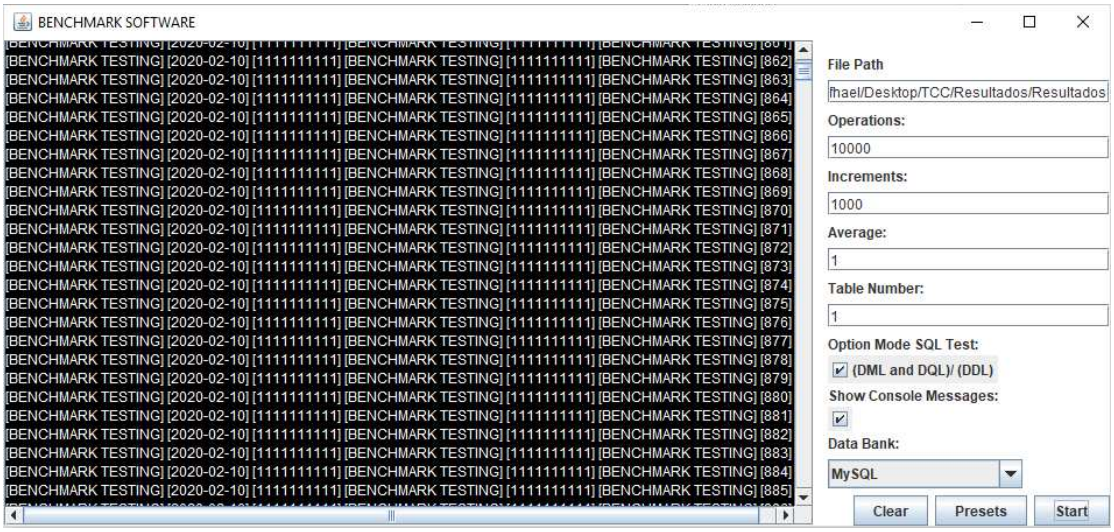


Figura 22 - Visualização de Dados da Função de Seleção

Fonte – Autoria Própria

Conforme pode ser visualizado na figura 22, o console da interface gráfica mostra todos os valores dos atributos de uma tabela que foram obtidos pelo método de seleção, cada valor apresentado na figura está entre colchetes no console da interface gráfica.

3.5.4 Interface Gráfica

A interface gráfica, vista nas figuras 19 e 22, foi desenvolvida com o objetivo de facilitar a visualização das informações geradas tanto pelo *software* de *benchmark* quanto as geradas pelos bancos de dados e também facilitar o estabelecimento dos parâmetros de entrada, a interface dispõe de um menu conforme a figura a seguir:

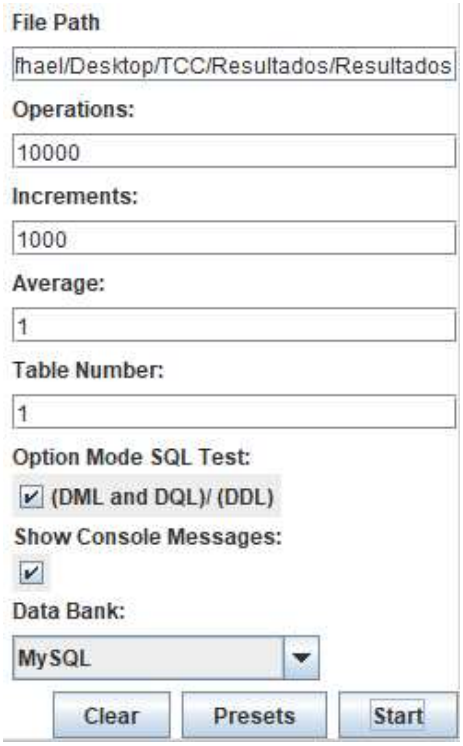


Figura 23 - Menu da Interface Gráfica do Projeto

Fonte: Autoria Própria

O menu da interface gráfica do projeto possui as opções de atribuir o local que será salvo os resultados das análises, selecionar a quantidade de operações máxima que o algoritmo fará, o incremento dessas operações a cada iteração, a quantidade de vezes que o algoritmo fará o processo, a quantidade de tabelas de testes, as linguagens SQL que serão testadas e a os bancos de dados que serão analisados.

Antes da opção de escolha dos bancos de dados existe um botão que permite o usuário ver todas as operações que estão sendo realizadas pelo *software* no momento da análise, caso não seja necessário visualizar essas operações, o console da interface gráfica somente mostrará uma mensagem ao fim de cada iteração do

algoritmo até o término da análise, quando este mostrará os arquivos sendo criados e as informações dos resultados sendo armazenadas e salvas.

O botão descrito como Option Mode seleciona os testes realizados com as operações de inserção, seleção, modificação e exclusão ou as operações de criação e exclusão de tabelas.

O botão de nome Presets seleciona diversas configurações para as variáveis do menu, essas configurações foram estabelecidas no *software* para executar testes com operações, tabelas e volumes de dados que eram recorrentemente selecionados

3.5.5 Algoritmo de Modificação

A função de modificação de valores muda o valor de um atributo em uma tabela a partir de uma cláusula *WHERE*.

O processo de modificação ocorre da seguinte forma: o *software* busca uma tabela e o atributo que deve ser modificado e a função executa o comando SQL até o término estipulado por um número máximo de modificações determinado antes do início da análise.

O algoritmo também leva em consideração o número de inserções realizadas na hora da seleção do identificador de um conjunto de informações. Além disso, o *software* modifica um atributo segundo o seu tipo para novos valores de acordo com a tabela 4:

Tabela 4 - Tabela de Modificação de Valores

Tipo	Valores
INT	10101010
FLOAT	333333333
VARCHAR	UPDATE TEST
DATE	2020-16-01

Fonte: Autoria Própria

Os valores contidos na tabela 4 são atribuídos às variáveis contidas em tabelas quando submetidas ao método de modificação. Foram atribuídos valores que diferenciassem os atributos modificados para que pudessem ser identificados mais facilmente em uma verificação visual do correto funcionamento do algoritmo.

3.5.6 Algoritmo de Exclusão

O algoritmo de exclusão das informações das tabelas em um banco de dados atua semelhantemente ao algoritmo de inserção. O sistema busca por informações em toda a base de dados, excluindo sequencialmente registros de todas as tabelas presentes.

O algoritmo contém um número máximo de exclusões que deve ser sempre menor ou igual ao número de inserções que foram feitas. Além disso o próprio *software* deve manter um registro de todos os identificadores em uma tabela que foram excluídos para que não selecione um que já não existe e assim provoque uma resposta negativa do banco de dados aumentando assim o seu tempo de execução.

O algoritmo foi implementado com a cláusula *DELETE ON CASCADE*, esta promove a exclusão de todas as informações que estão em outras tabelas e possuem algum tipo de relação com o dado que está sendo excluído naquele momento. Esta cláusula é uma propriedade dos bancos de dados relacionais, logo não foi necessário implementar nenhum sistema de busca por informações que tivessem relação com a informação excluída de uma determinada tabela, visto que o próprio banco de dados já realiza este procedimento. O algoritmo foi desenvolvido com esta cláusula para possíveis testes envolvendo o modelo E/R com chaves estrangeiras ligando as tabelas em uma base de dados.

3.6 Sistema de Contagem de Tempo

O sistema de contagem de tempo funciona por meio de uma classe Java que possui funções que atuam em paralelo a execução do *software* principal, estas são chamadas de *threads*.

A contagem de tempo é ligada em determinados momentos da execução do programa principal, quando o algoritmo executa as operações no banco de dados, e é desligada após a conclusão dessas operações.

O tempo contabilizado de uma função é obtido em nanosegundos, podendo ser convertido por meio de outras funções em tempos como: milissegundos, segundos e horas.

3.7 Sistemas de Armazenamento de Resultados

Neste tópico são apresentados os sistemas que armazenam as informações provenientes dos testes.

3.7.1 Armazenamento em Arquivo Texto

O algoritmo para o armazenamento dos resultados consiste em um sistema que cria um arquivo texto e copia as informações obtidas dos testes para o mesmo.

Este algoritmo foi criado devido a necessidade de armazenar a informação de forma a poder ser visualizada e resgatada facilmente.

O sistema salva as informações a cada ciclo de testes para evitar que algum problema externo, como falta de energia por exemplo, cause a perda de todas as métricas.

3.7.2 Armazenamento no Excel

O armazenamento das informações em um arquivo Excel foi desenvolvido para o processo de criação dos gráficos na plataforma. A automatização da transferência das informações para as planilhas evita que haja erros de desatenção, que poderiam causar consequentemente erros nos gráficos, e torna mais rápida a obtenção dos resultados uma vez que não é necessário a escrita dos mesmos nas planilhas.

O algoritmo salva todas as informações registradas ao longo de toda a análise por meio de uma matriz de objetos que mapeia cada célula de um arquivo em Excel, colocando as informações obtidas durante a análise nas células mapeadas. Logo só é possível salvar as informações em um arquivo Excel ao final do processo de análise, visto que é necessária a construção dessa matriz de objetos para o mapeamento.

4.0 RESULTADOS OBTIDOS

Este capítulo apresenta as configurações do *hardware*, dos bancos de dados e dos testes, assim como o resultado comparativo dos *softwares* que foram analisados.

4.1 Ambiente de Simulação

A seguir encontra-se uma tabela com as especificações do computador que foi utilizado para executar os testes:

Tabela 5 - Configuração do Computador de Teste

Processador	Intel Core i7-4770 3400 GHz
Memoria	8 Gb
Sistema Operacional	Windows 10 - 64 bits
Espaço em Disco	1 Tb

Fonte: Autoria Própria

Os testes foram realizados em um computador *desktop*, todos os bancos de dados foram executados em *localhost*, as simulações foram realizadas somente com a verificação e visualização das operações por meio do próprio software de *benchmark*.

4.2 Versões dos Bancos de Dados

Foram utilizadas as versões mais atuais disponíveis para cada banco de dados, a seguir encontra-se uma tabela com as versões utilizadas:

Tabela 6 - Tabela de versões dos bancos de dados utilizados

Banco de dados	Versão
MySQL	8.0.19
PostgreSQL	12.2
FireBird	2.5

Fonte: Autoria Própria

4.3 Configuração das Variáveis Internas dos Bancos

Todo o sistema gerenciador possui variáveis internas que podem ser alteradas como: quantidade de memória para ser usada por tabelas e indexadores, quantidade de *threads* de leitura e escrita de informação, quantidade de threads concorrentes, entre outras.

Neste projeto foi optado por deixar a configuração padrão destas variáveis em todos os sistemas, visto que o intuito não era beneficiar nem prejudicar nenhum SGBD.

4.4 Procedimento de Execução dos Testes

Para os testes com as operações CRUD, o algoritmo executou as operações sequencialmente: primeiro aplicou-se a operação de inserção de valores, seguido de consultas, atualizações e exclusões na base de dados dos bancos.

Os testes foram realizados em sequências de pequenos à grandes volumes de dados, de 50 a 500 informações e de 50 mil a 500 mil registros respectivamente, usando a base de dados que contém a tabela descrita na figura 16.

Para pequenos volumes de dados o algoritmo foi executado 250 vezes para produzir a média dos valores obtidos e para os grandes volumes de dados, o algoritmo foi executado 50 vezes obtendo-se também a médias destes resultados.

Os testes com as operações de criação e remoção de tabelas foram realizados com uma, cem e mil operações, as tabelas descritas na figura 18. A média foi retirada também de 50 amostras assim como as operações da linguagem DML e DQL.

Os valores escolhidos e utilizados para a realização da comparação, ou seja, os intervalos dos volumes de dados bem como as quantidades de tabelas para a criação foram escolhidos devido a inúmeros testes que foram realizados, sendo selecionados os parâmetros que deram o melhor resultado.

4.5 Tempo de Realização dos Testes

O tempo total da execução de cada teste foi de 2 horas e 20 minutos para os das operações CRUD com pequenos volumes de dados, 18 horas e 36 minutos para

a realização dos testes com grandes volumes de informação e 12 horas e 26 minutos para os testes com as operações de criação e remoção de tabelas.

4.6 Realização dos Testes

Esta seção trata a respeito dos resultados obtidos com os testes das operações SQL apresentando gráficos gerados a partir dos mesmos.

4.6.1 Operação de Inserção

Os testes das operações de inserção foram iniciados com um pequeno volume de dados, destes testes foram obtidos os gráficos que podem ser visualizados pela figura 24:

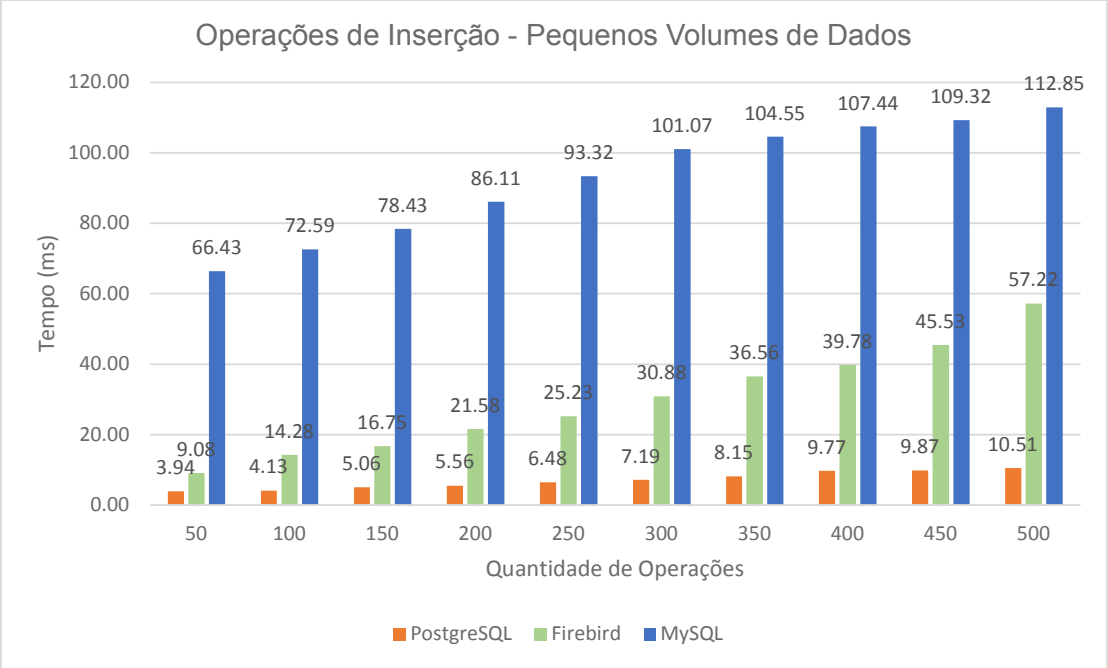


Figura 24 – Operações de Inserção com Pequeno Volume de Dados

Fonte: Autoria Própria

Conforme é possível observar a partir do gráfico da figura 24 o tempo de inserção cresce à medida que aumenta o número de operações e também o próprio volume de dados que é inserido durante o processo. Dessa forma, quanto maior a

quantidade de operações, mais difícil é para um banco de dados inserir valores em uma tabela.

Para a quantidade de operações apresentada na figura 24, o banco de dados Firebird conseguiu realizar as inserções mais rápido que o MySQL, a diferença dos tempos de inserção para o primeiro volume de dados foi de 57,35ms.

Subtraindo os tempos de cada lote de dados pelo seu anterior e retirando a média desses valores, verificou-se que o aumento de tempo da operação a cada lote inserido era em média 5,3ms para o Firebird e 5,16ms para o MySQL. Como visto no gráfico da figura 24, o Firebird ao final do processo realizou seu último lote de operações com uma diferença de 55,64ms em relação ao MySQL.

Os dois bancos de dados comparados ao PostgreSQL realizaram suas operações de forma mais lenta, a diferença inicial entre este sistema foi de 62,48ms em relação ao MySQL e 5,14ms em relação ao Firebird. O tempo de inserção entre os lotes de dados também foi o menor dentre os três sistemas, em média 0,73ms, realizando o último com uma diferença de 102,34ms em relação ao MySQL e 46,7ms em relação ao Firebird.

A seguir na figura 25 é apresentado o resultado dos testes realizados com grandes volumes de dados:

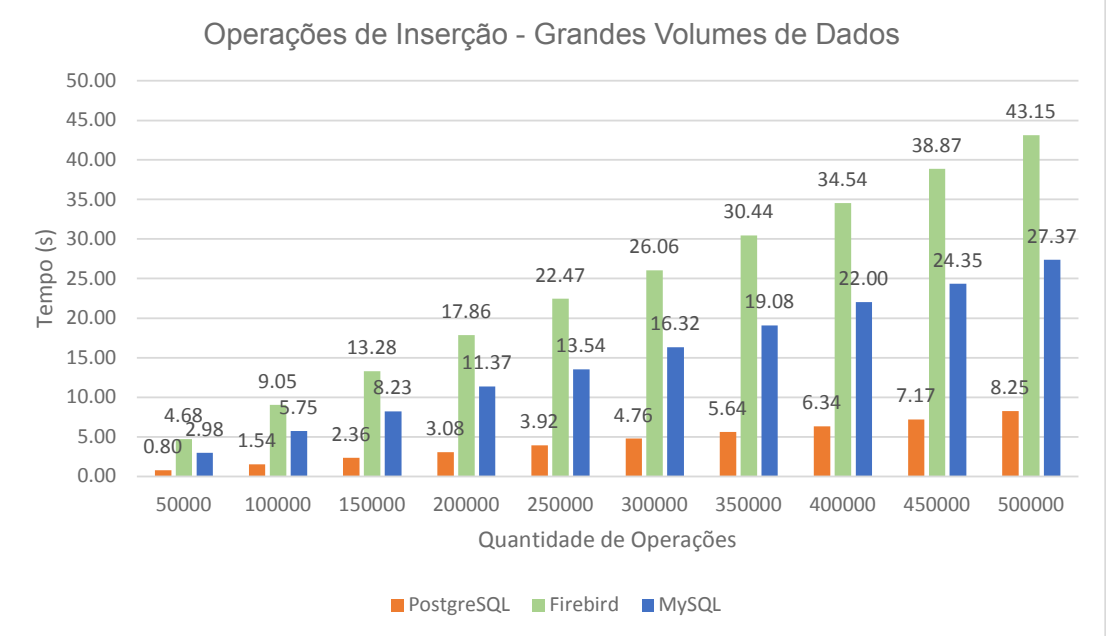


Figura 25 – Operações de Inserção com Grandes Volumes de Dados

Fonte: Autoria Própria

Os testes realizados com grandes volumes de dados mostraram que conforme o aumento da quantidade de operações, o banco de dados Firebird possuiu maior dificuldade na operação. Para o primeiro lote de informações, a diferença de inserção em relação ao MySQL foi de 1,7s e em relação ao PostgreSQL foi de 3,68s.

A diferença de tempo entre os lotes de operações foi em média 4,27s para o Firebird contra 2,71s do MySQL. Ao final do experimento o primeiro atingiu o patamar de 43,15s no último lote de operações e o segundo 27,37s.

O PostgreSQL obteve melhores resultados que os outros dois bancos de dados nos testes com grandes volumes de dados, a diferença de inserção entre os lotes de operações foi em média 0,83s, O PostgreSQL iniciou os testes com uma diferença de 2,18s em relação ao MySQL e 3,89s em relação ao Firebird e terminou o experimento realizando o último lote com uma diferença de 19,12s em relação ao primeiro e 34,89s em relação ao segundo.

4.6.2 Operações de Seleção

O gráfico das operações de seleção para pequenos volumes de dados é apresentado por meio da figura 26:

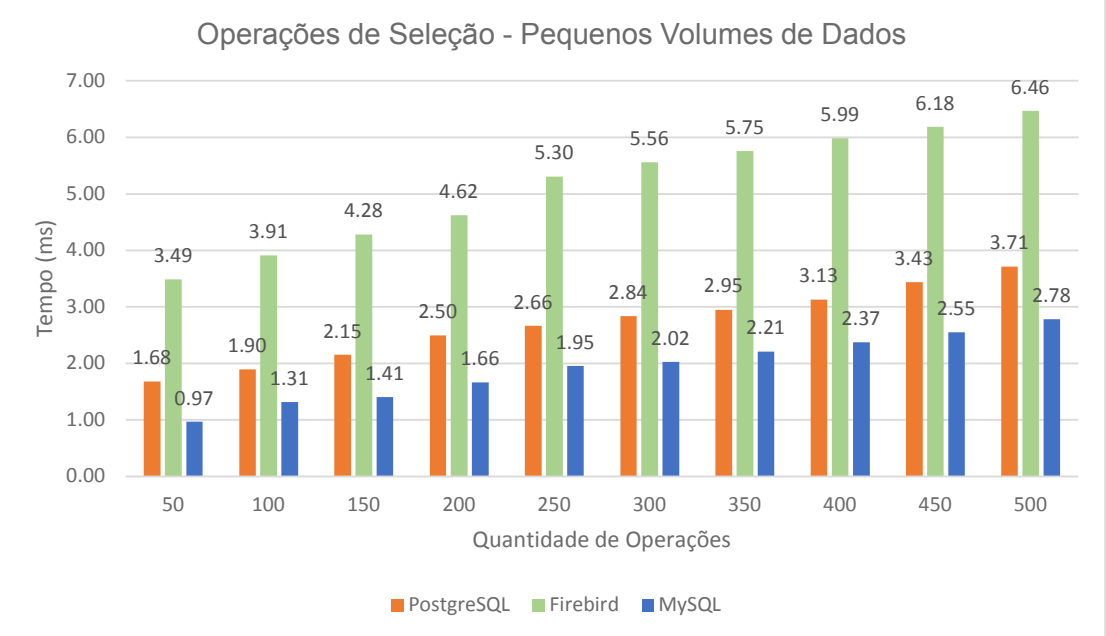


Figura 26 - Operações de Seleção com Pequenos Volumes de Dados

Fonte: Autoria Própria

Para as quantidades de operações apresentadas na figura 26, O banco de dados Firebird foi o sistema que selecionou as informações de forma mais lenta. No início dos testes, a diferença entre ele e o MySQL foi de 2,52ms e 1,81ms em relação ao PostgreSQL, A diferença de tempo da operação entre os lotes de comandos de seleção foi em média 0,33ms.

O Firebird realizou o último ciclo de seleções em 6,46ms, uma diferença de 2,75ms em relação ao PostgreSQL e 3,38ms em relação ao MySQL.

Em relação ao desempenho do PostgreSQL e o MySQL, o primeiro demorou mais tempo para realizar as operações, 0,71ms a mais no primeiro lote de operações aumentando essa diferença conforme o andamento do experimento, o sistema PostgreSQL finalizou o ciclo de testes selecionando o último lote de operações em 3,71ms contra 2,78ms do MySQL, uma diferença de 0,93ms entre os sistemas.

Realizando os testes para grandes volumes de dados, obteve-se o gráfico que pode ser visualizado por meio da figura 27:

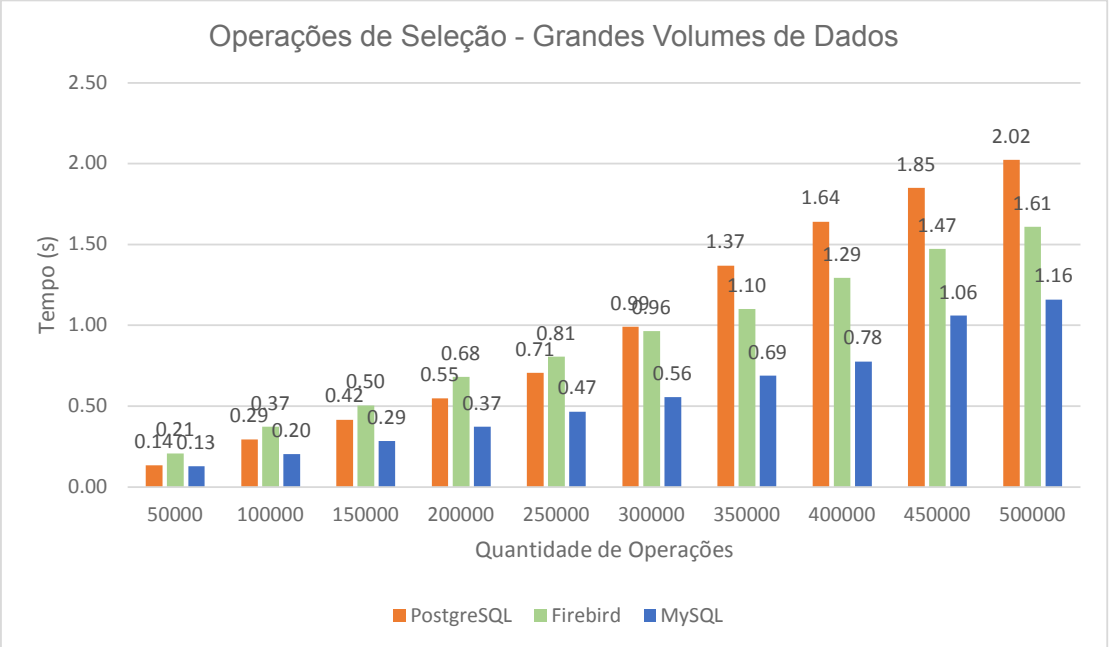


Figura 27 - Operações de Seleção com Grandes Volume de Dados

Fonte: Autoria Própria

O gráfico da figura 27 representa as operações de seleção realizadas em grandes volumes de dados, o Firebird iniciou o experimento com tempos de seleção

mais elevados que os demais bancos, no primeiro volume de dados a diferença entre este banco e os demais é de 0,07s em relação ao PostgreSQL e 0,08s em relação ao MySQL.

A diferença de tempo entre os lotes de operações de seleção foi em média 0,16s para o Firebird em relação a 0,21s do PostgreSQL e conforme o aumento do volume de dados selecionado pelos bancos, o PostgreSQL começou a demorar mais para selecionar as informações do que o Firebird, chegando ao final do ciclo de operações a realizar o último lote de seleções em 2,2s contra 1,61s do Firebird.

O MySQL desde o primeiro lote de operações obteve melhores resultados do que os outros dois sistemas, que se manteve ao longo da análise. A diferença de tempo entre os lotes de seleção para este sistema foi em média 0,11s.

4.6.3 Operações de Modificação

O gráfico que contém as informações dos testes de modificação para pequenos volumes de dados pode ser visualizado por meio da figura 28:

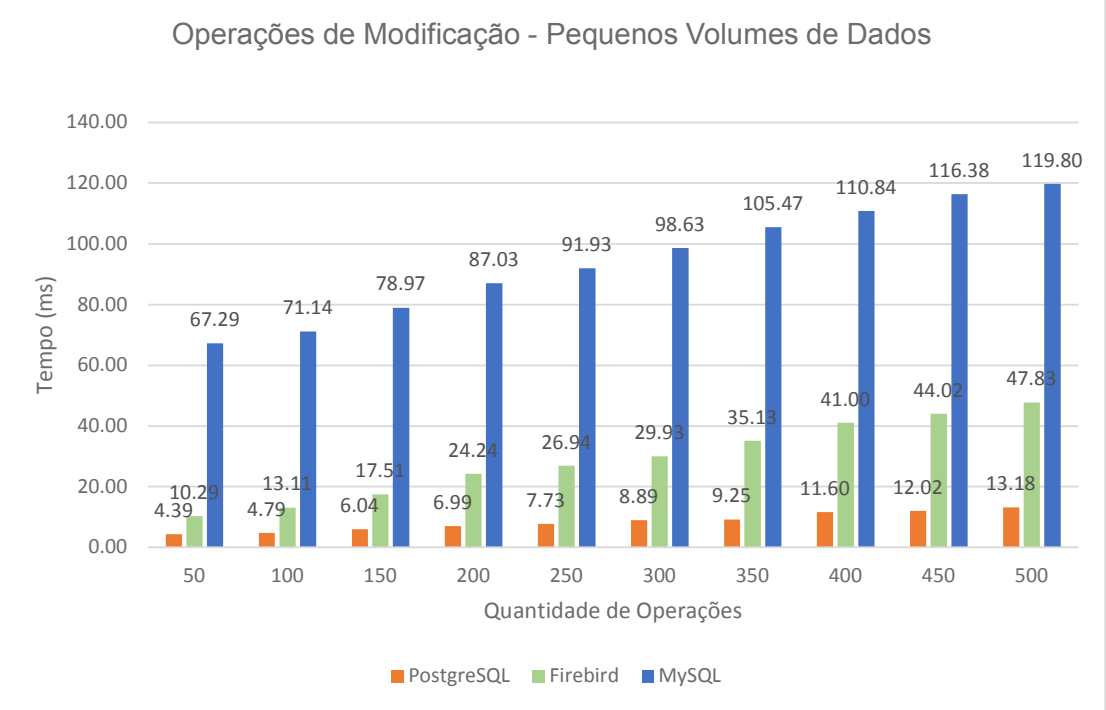


Figura 28 - Operações de Modificação com Pequenos Volume de Dados

Fonte: Autoria Própria

O MySQL iniciou o ciclo de operações realizando o primeiro lote em 67,29ms, uma diferença de tempo 62,90ms em relação ao PostgreSQL e 57,01ms em relação ao Firebird. A diferença de tempo entre os lotes da operação de modificação foi em média de 5,83ms e o sistema completou o último ciclo de operações em 119,8ms.

Os outros dois bancos de dados, assim como nos testes de inserção, inicializaram o ciclo de operações com tempos menores do que o MySQL. A diferença de tempo entre os lotes de operações foi em média 4,17ms para o Firebird e 0,98ms para o PostgreSQL, levando o primeiro a realizar o último lote de operações em 47,83ms e o segundo a executar em 13,18ms.

A diferença apresentada entre o MySQL e os outros bancos de dados para a último lote de operações foi de 106,62ms em relação ao PostgreSQL e 71,97ms em relação ao Firebird.

Os testes realizados com grandes volumes de dados podem ser visualizados por meio do gráfico da figura 29:

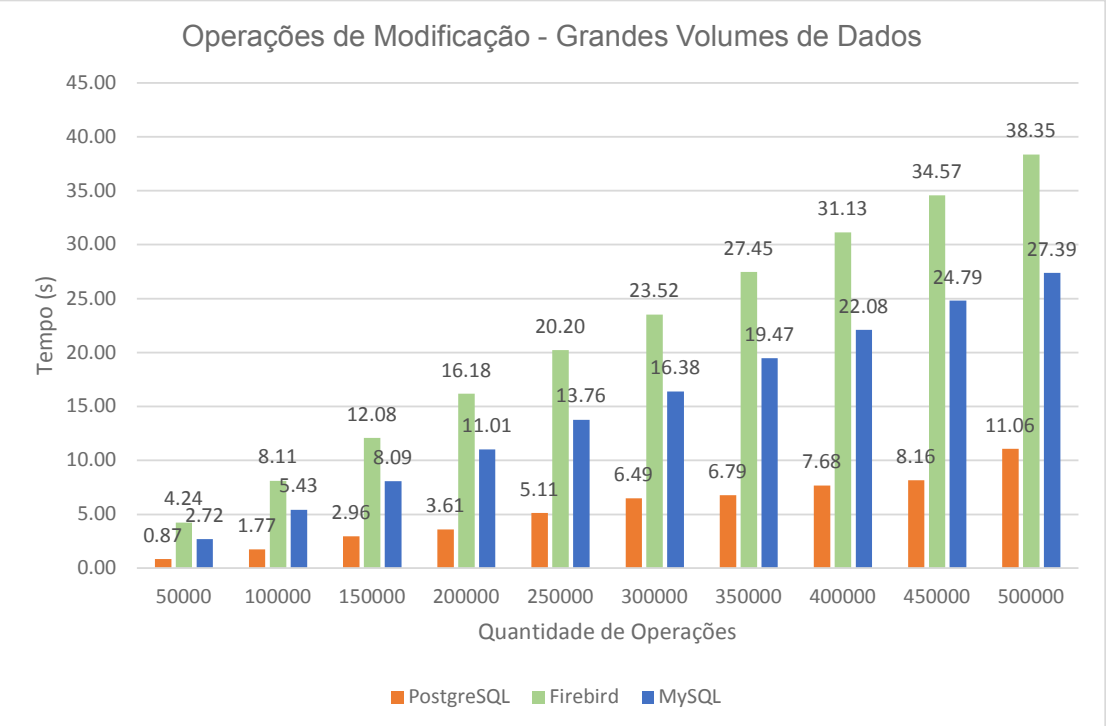


Figura 29 - Operações de Modificação com Grandes Volume de Dados

Fonte: Autoria Própria

Percebe-se que o Firebird assim como nos testes de inserção realizou o primeiro lote de operações com tempo superior aos outros dois bancos de dados, a diferença entre ele e os demais para o primeiro volume foi de 3,37s em relação ao PostgreSQL e 1,52s em relação ao MySQL.

A diferença de tempo entre os lotes de comandos de modificação foi em média 3,79s para o Firebird, 2,74s para o MySQL e 1,13s para o PostgreSQL. O Firebird realizou o último lote de operações em 38,35s, caracterizando uma diferença entre o último lote modificado pelos outros dois bancos de dados de 10,97s em relação ao MySQL e 27,29s em relação ao PostgreSQL.

4.6.4 Operações de Exclusão

A seguir é apresentado o gráfico que contém os resultados dos testes de exclusão para pequenos volumes de dados:

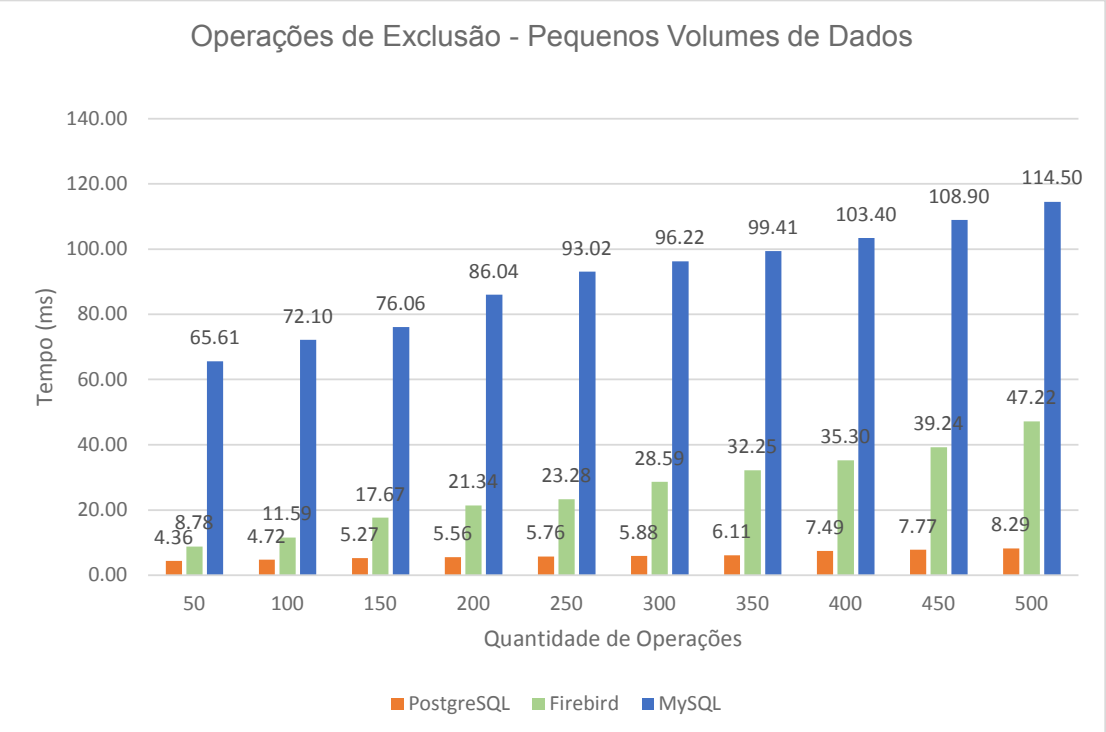


Figura 30 - Operações de Exclusão com Pequenos Volumes de Dados

Fonte: Autoria Própria

Para as quantidades de operações apresentadas na figura 30, o MySQL demorou mais para excluir as informações da base de dados do que os outros bancos. No começo da análise o MySQL realizou o primeiro lote de operações em 66,61ms, uma diferença de 56,82ms em relação ao Firebird e 60,88ms em relação ao PostgreSQL. A diferença entre os lotes da operação de exclusão foi em média 5,43ms para o MySQL, 4,27ms do Firebird e 0,44ms do PostgreSQL

O MySQL realizou o último lote de operações em 114,5ms, uma diferença de 67,28ms em relação ao Firebird e 106,21ms em relação ao PostgreSQL.

O gráfico que contém os resultados dos testes realizados em grandes volumes de dados pode ser visualizado por meio da figura 31:

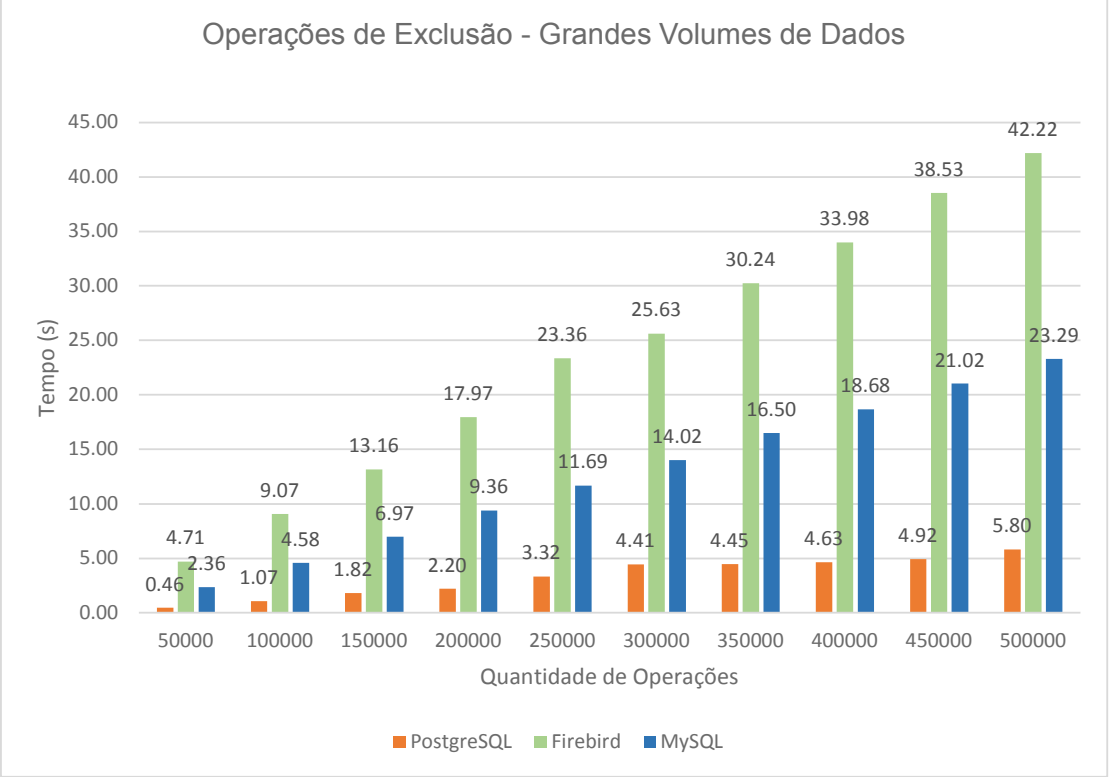


Figura 31 - Operações de Exclusão com Grandes Volumes de Dados

Fonte: Autoria Própria

Como pode ser observado na figura 31, com o primeiro volume de dados apresentado, o Firebird obteve tempos maiores de exclusão que os outros dois bancos de dados, com uma diferença no primeiro lote de 4,25s em relação ao PostgreSQL e 2,34s em relação o MySQL.

A diferença de tempo entre os lotes para as operações de exclusão foi em média 4,17s para o Firebird, este realizou o último lote de operação em 42,22s, uma diferença de 36,42s em relação ao PostgreSQL e 18,94s em relação ao MySQL.

O PostgreSQL obteve os menores tempos de exclusão dentre os bancos de dados analisados. Excluindo o primeiro lote de operações em 0,46s, o aumento de tempo a cada lote excluído foi em média 0,59s e ao final do experimento, ele executou o último lote de exclusão em 5,8s, uma diferença de 17,49s em relação ao MySQL.

4.6.5 Consumo de Tempo entre as Operações nos Bancos de Dados

O gráfico do consumo de tempo de cada operação para a análise realizada no banco de dados PostgreSQL pode ser visualizada por meio da figura 32:

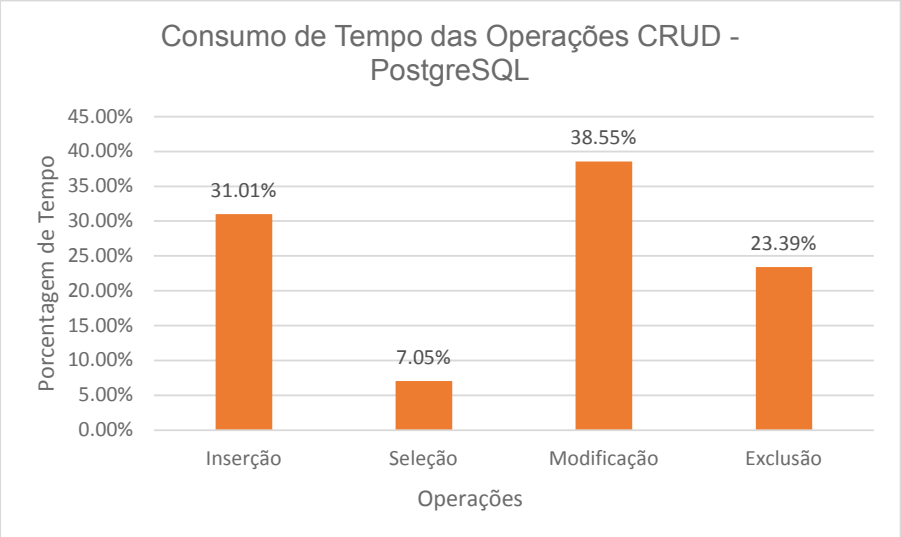


Figura 32 - Consumo de Tempo das Operações CRUD – PostgreSQL

Fonte: Autoria Própria

Conforme é possível observar pelas informações da figura 32, a operação que demandou maior tempo do sistema foi a operação de modificação com 38,55% do tempo gasto, caracterizando assim a operação de maior custo para o sistema. Em seguida a operação de inserção que ocupou 31,01% do tempo, a exclusão com 23,39% e operação de seleção com 7,05%.

A figura 32 mostra o gráfico que contém a porcentagem de tempo de consumo de cada operação na análise realizada no banco de dados Firebird:

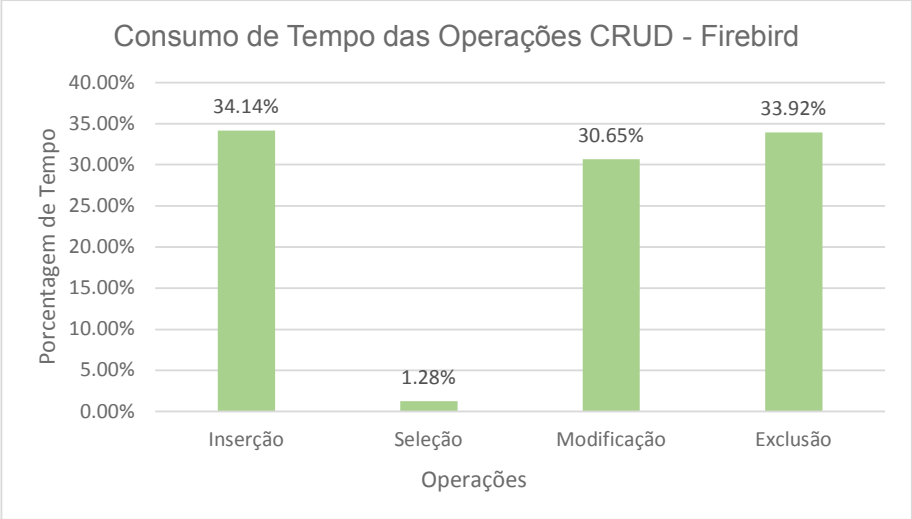


Figura 33 - Consumo de Tempo das Operações CRUD – Firebird

Fonte: Autoria Própria

Observando a figura 33, nota-se que a operação que deteve a maior quantidade de tempo para a execução por parte do sistema foi a operação de inserção com 34,14% do tempo gasto, seguida da operação de exclusão com 33,92%, modificação com 30,65% e a seleção com 1,28%.

A figura a seguir mostra o consumo de tempo de cada operação CRUD para o banco de dados MySQL:

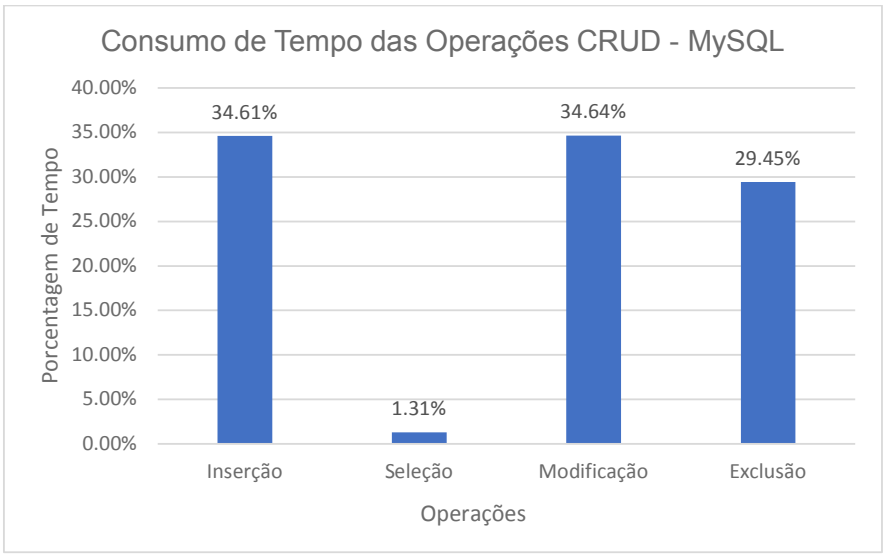


Figura 34 - Consumo de Tempo das Operações CRUD - MySQL

Fonte: Autoria Própria

Observando a figura 33, nota-se que a operação de maior custo para o MySQL foi a operação de modificação com 34,64% do tempo gasto, em seguida a operação de inserção com 34,61, exclusão com 29,45% e seleção com 1,31%.

4.6.6 Testes de Criação de Tabelas

Foram realizados testes com criações de tabelas nas bases de dados de cada banco, os resultados podem ser visualizados por meio da figura 35:

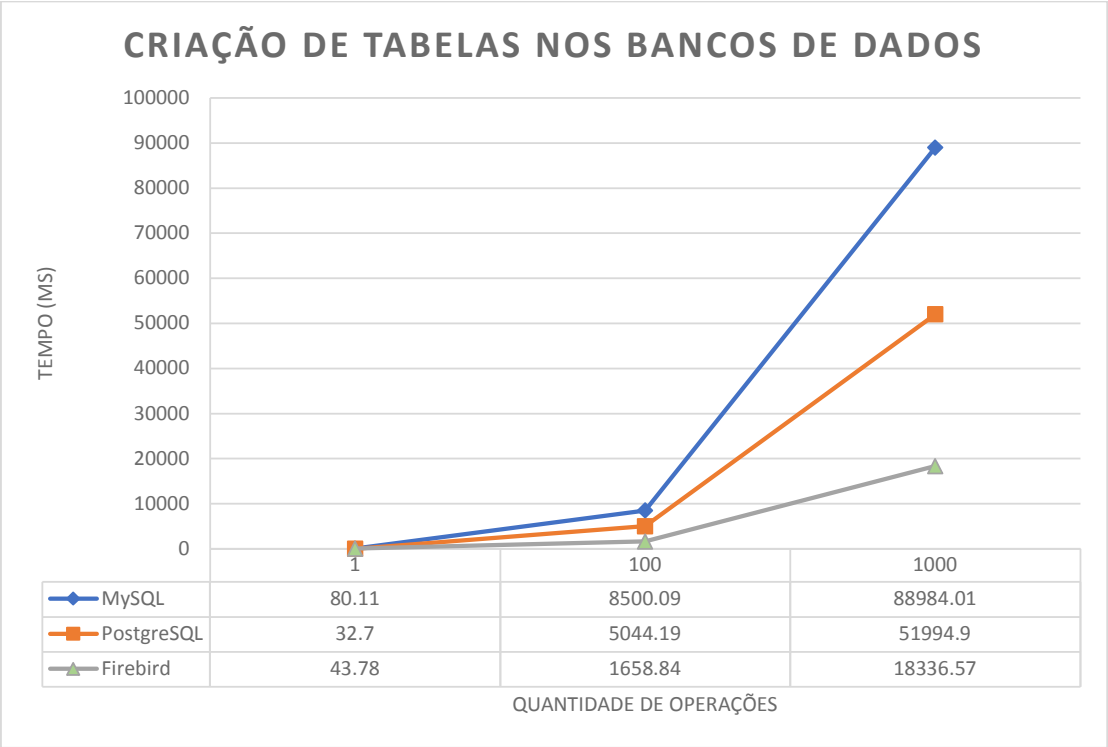


Figura 35 - Gráfico do Tempo de Criação de Tabelas nos Bancos de Dados

Fonte: Autoria Própria

A quantidade de operações da figura 35 reflete a quantidade de tabelas que o sistema produz a partir do algoritmo de geração de bases de dados, pelo gráfico é possível notar que o tempo de criação de uma tabela pelo MySQL é maior que os tempos dos outros bancos de dados.

Na criação de uma tabela, o MySQL demorou 80,11ms, uma diferença de 47,41ms em relação ao PostgreSQL e 36,33ms em relação ao Firebird. Conforme o aumento da quantidade de operações, o MySQL passou a ter maior dificuldade em

criar tabelas em sua base de dados, na criação de cem tabelas, a diferença de tempo em relação ao PostgreSQL foi de 3,46s e 6,84s em relação ao Firebird.

Ao final do ciclo de testes, na criação de mil tabelas, o MySQL completou as operações em 88,98s, com uma diferença de 36,99s em relação ao PostgreSQL e 70,75s em relação ao Firebird.

O banco de dados PostgreSQL obteve melhor desempenho que o Firebird na criação de uma tabela, mas conforme o aumento da quantidade de operações, o Firebird passou a criar as tabelas de forma mais rápida criando ao final da análise mil tabelas em 18,34s, 33,66s mais rápido que o PostgreSQL.

4.6.7 Testes de Exclusão de Tabelas

Foram realizados testes com exclusões de tabelas nas bases de dados dos bancos, a mesma quantidade de operações do processo de criação de tabelas foi aplicada à exclusão de tabelas, visto que a função de exclusão depende que haja as tabelas criadas pela função de criação de tabelas. Os tempos de exclusão de tabelas nas bases de dados dos bancos pode ser visto pelo gráfico da figura 36:

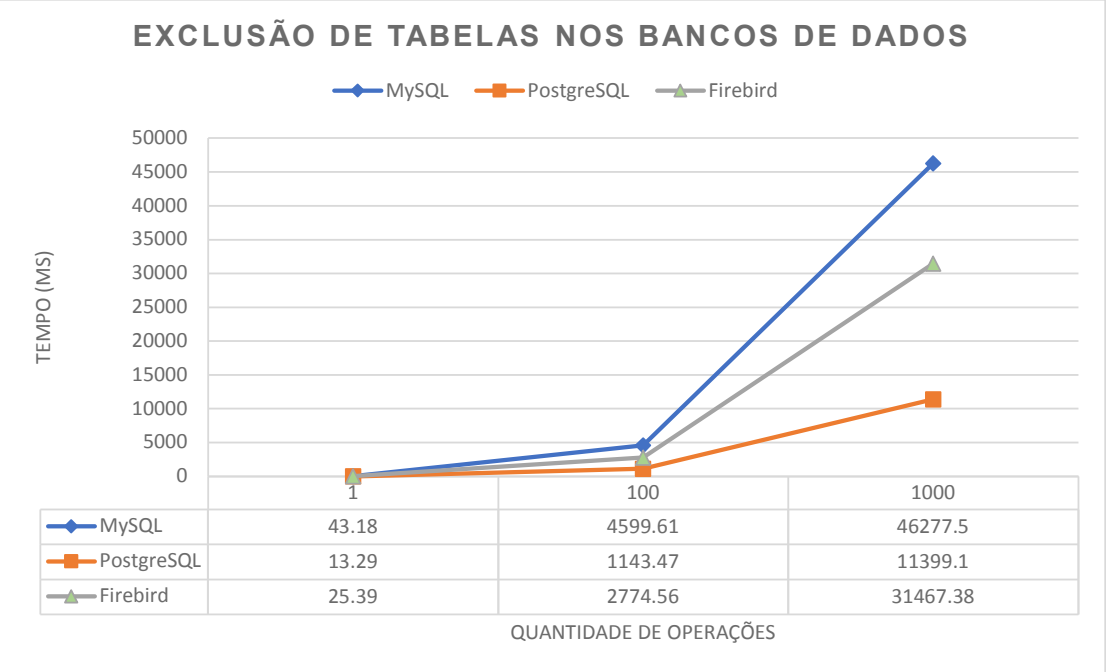


Figura 36 - Gráfico do Tempo de Exclusão de Tabelas nos Bancos de Dados

Fonte: Autoria Própria

A partir da análise do gráfico da figura 36 é possível notar que assim como na criação de tabelas, o MySQL demorou mais tempo para excluir as tabelas em sua base de dados. No início dos testes a diferença entre a exclusão de uma tabela do MySQL foi de 31,05ms em relação ao PostgreSQL e 28,41ms em relação ao Firebird.

Conforme o aumento da quantidade de operações, o MySQL passou a ter maior dificuldade na operação, em uma base de dados contendo 100 tabelas, o MySQL realizou a operação em 4,06s contra 1,4s do PostgreSQL e do Firebird, estes mantiveram tempos semelhantes para as quantidades de operações de exclusão de uma e cem tabelas, contudo, para a quantidade de operações de exclusão de mil tabelas, ocorreu o aumento do tempo de exclusão do Firebird superando os tempos do PostgreSQL, comportamento que não foi detectado na operação de criação de tabelas, finalizando o ciclo de testes com uma diferença de 22,4s em relação ao mesmo.

O MySQL finalizou o ciclo de testes excluindo mil tabelas em 45s, com uma diferença de 33,5s em relação ao PostgreSQL e 11,1s em relação ao Firebird.

5 CONCLUSÕES

Os testes realizados nas linguagens de manipulação de dados, ou seja, as operações de inserção, modificação e exclusão entre o PostgreSQL e o MySQL mostraram que o primeiro teve resultados superiores ao segundo desde pequenos a grandes volumes de dados. Os resultados mostraram que para essas operações, o PostgreSQL foi o banco que menos sofreu com o aumento do volume de dados. Nos testes realizados com a linguagem de consulta de dados, as operações de seleção, o PostgreSQL apresentou piores resultados do que o MySQL.

As linguagens de manipulação de dados apresentaram um padrão no qual para pequenos volumes de dados, o sistema Firebird obteve melhores resultados que o MySQL, conforme houve o aumento da quantidade de operações, o sistema deparou-se com volumes de dados cada vez maiores, com isso ocorreu o aumento do tempo do sistema em executar as operações. Entretanto na linguagem de consulta de dados, ocorreu que para pequenos volumes de informação, o Firebird demorou mais que os outros bancos de dados para realizar as operações, contudo, conforme o aumento do volume de dados, houve o aumento do tempo de execução por parte do PostgreSQL, que terminou a análise com o tempo mais elevado nesta operação.

A análise geral para as operações das sublinguagens SQL DML e DQL apontou que o PostgreSQL obteve melhores resultados que os outros dois bancos de dados, os tempos das operações da sublinguagem SQL DML do PostgreSQL foram significativamente menores que os demais, além disso, para essas operações esse banco de dados foi o que menos teve interferência das quantidades de informações inseridas, isso pode ser notado por meio do cálculo feito da diferença de tempo entre um lote de comandos e seu antecessor. Na seleção de dados, realmente ocorreu o aumento do tempo do PostgreSQL frente aos volumes de dados inseridos, porém a diferença de tempo em relação aos outros sistemas não foi tão significativa como foi visto no decorrer da análise até o último volume de dados de 500 mil informações, neste a diferença de tempo do PostgreSQL para o MySQL, banco que obteve o melhor resultado nessa operação, é de menos de 1 segundo.

A análise geral das operações CRUD mostrou que cada operação possui um custo convertido em tempo de execução das tarefas. A operação de maior custo foi a de modificação, seguido da inserção, exclusão e seleção.

Analisando estas operações em cada banco isoladamente, conclui-se que cada um deles teve uma operação que representou maior custo. Para o MySQL e o PostgreSQL, a operação de maior custo foi a modificação, no Firebird, a operação de maior custo foi a inserção. Todos eles tiveram a seleção como a operação de menor custo.

Em relação a sublinguagem SQL DDL, o banco de dados Firebird obteve os melhores resultados na operação de criação de tabelas, sendo 2 vezes mais rápido que o PostgreSQL e quase 4 vezes mais rápido que o MySQL. Nas operações de exclusão de tabelas ocorreu que o PostgreSQL obteve os melhores resultados, sendo quase 3 vezes mais rápido que o Firebird e 4 vezes que o MySQL.

A análise mostrou que para os bancos de dados MySQL e PostgreSQL, o custo da operação de criação de tabelas é maior do que o custo de remoção de tabelas. Para o banco de dados Firebird ocorre o oposto, o custo das operações de remoção de tabelas é maior do que o custo de criação das mesmas.

A quantidade de iterações que o algoritmo fez para a obtenção das médias dos resultados produziu valores que puderam transmitir diferenças de desempenho, contudo, principalmente para os pequenos volumes de dados, em alguns lotes de dados e seu subsequente percebe-se que não houve muita diferença no aumento de tempo mesmo com uma média produzida de 250 iterações.

As quantidades de operações atribuídas no experimento produziram volumes de dados que provocaram impacto sobre os tempos das operações dos bancos de dados, contudo, para grandes volumes de dados, a análise tornou-se muito demorada em virtude da quantidade de iterações que foram necessárias para a obtenção das métricas.

O trabalho realizado procurou criar uma ferramenta para analisar o desempenho de bancos de dados relacionais, estabelecendo uma métrica que pudesse apresentar resultados visíveis para serem comparados entre os sistemas gerenciadores. Durante esse processo verificou-se que a tarefa em si era de grande complexidade e que existiam organizações especializadas em construir *softwares* para essa finalidade. A solução para apresentar resultados relevantes foi diminuir o problema a um nível que pudesse ser tratado com o recurso de *hardware* disponível.

Este trabalho apresenta resultados de três sistemas nas condições de suas variáveis internas no formato padrão. Contudo durante a realização do trabalho descobriu-se que as variáveis internas das *engines* onde são desenvolvidos esses

softwares podem ser modificadas como: tamanho do *buffer* de armazenamento de dados, quantidade de *threads* de leitura e escrita, número de *threads* concorrentes, entre outras. Em trabalhos relacionados a benchmark, em nenhum momento foi encontrado relatos de que essas variáveis foram alteradas para a execução dos testes. Contudo, houve o questionamento se configurações diferentes em um determinado banco de dados poderia beneficia-lo de forma a apresentar resultados que superassem os outros sistemas. Além disso, é conhecido que *hardwares* diferentes que executam tanto o *software* de *benchmark* quanto os bancos de dados produzem resultados quantitativos diferentes não variando o resultado geral, mas fornecendo diferenças em métricas.

Em virtude do que foi apresentado, a maior contribuição que este trabalho pode oferecer é relatar ao leitor que antes de ser avaliado o desempenho de um banco de dados, é necessário avaliar em que condições de *hardware* e *software* foram realizados os testes de desempenho deste banco. Conforme será visto na seção 5.1, as dificuldades de se desenvolver um software de *benchmark* estão intimamente ligadas a correta manipulação das operações que este precisa realizar nos bancos de dados, para isso é necessário que este tenha controle das funcionalidades dos sistemas.

Existiram dificuldades na realização deste trabalho, as quais são apresentadas no tópico a seguir:

5.1 Dificuldades e Limitações

A primeira dificuldade encontrada na realização deste projeto é a falta de uma metodologia específica para a aquisição dos resultados que foram apresentados, como estas tecnologias evoluem constantemente, é muito difícil encontrar um livro que apresente passo a passo uma metodologia eficaz com os mesmos sistemas gerenciadores e ferramentas de programação utilizados, dessa forma, é necessário reunir as informações que estão contidas no estado da arte e adapta-las ao projeto.

Os artigos relacionados à *benchmark* fornecem análises que servem como parâmetro para a realização de trabalhos acadêmicos, entretanto, a variação do volume de dados encontradas nos artigos não pôde ser utilizada nesse projeto. Os artigos mostram variações de volumes de dados exponenciais com grandes quantidades de iterações, por exemplo: variações de mil, 10 mil e 100 mil informações

executando as operações CRUD 50 vezes, aumentando a precisão do cálculo. Entretanto, esse sistema não mostra a variação do tempo em relação ao volume de dados durante o processo, pois a diferença entre lote de operações e outro é muito grande. Nesse caso foi necessário aumentar o valor máximo das operações aplicadas, com isso obteve-se valores quantitativos que produziram diferenças que puderam ser vistas ao longo dos incrementos dos lotes de operações.

Um sistema que realiza uma análise envolvendo bancos de dados deve adequar-se a estes bancos de dados, mesmo trabalhando-se com paradigmas relacionais, cada sistema possui sua particularidade que somente é descoberta no decorrer do projeto e necessidades de adequação de código tornam-se fundamentais para desenvolver um *software* que proporcione o mesmo ambiente de teste em todos os sistemas analisados.

Os primeiros testes não foram bem-sucedidos, obtiveram-se grandes disparidades de tempos entre os bancos de dados, logo percebeu-se que o MySQL apresentava problemas nas operações de manipulação de dados, de forma a demorar muito para inserir um pequeno lote de informações.

Estes testes levantaram o questionamento a respeito da disparidade de tempo do sistema em relação aos demais e pesquisas tiveram que ser realizadas para entender o motivo.

Levantou-se a hipótese do *framework* JDBC utilizado para a comunicação com o MySQL estar desatualizado frente a uma nova versão do banco de dados, essa hipótese logo foi descartada diante das diferentes versões do mesmo banco de dados que foram testadas utilizando o mesmo *framework*.

Pesquisando a respeito do problema, foi constatado que as inserções lentas estavam sendo causadas devido ao fato da *engine* do MySQL, chamada InnoDB, não responder bem ao processamento por *query* de dados. Artigos de *benchmark* mostraram que o processamento por lotes de dados era o ideal para a realização de testes com este banco, modificações no projeto tiveram que ser realizadas e isso gerou perda de tempo e dos resultados obtidos antes da modificação.

O banco de dados Firebird possui uma particularidade que proporcionou problemas também com a validação por *query* de dados. Todos os bancos de dados possuem identificadores que servem para organizar as instruções que serão executadas, uma vez que o comando é concluído, este identificador é reiniciado podendo ser atribuído a outra instrução. O MySQL e o PostgreSQL executam a

reinicialização de forma automática, isso permite que um lote de 100 mil comandos de inserção seja executado sem a necessidade de reinicialização da conexão. Contudo como o Firebird não possui essa característica, quando este chega a um determinado limite de inserções, cerca de 60mil informações, o banco de dados trava a execução da operação e uma exceção Java é mostrada na tela, neste caso o número máximo atingido, sendo necessário fechar a conexão e inicializar uma nova para a inserção de mais valores. Executando as inserções por lotes de dados o problema foi resolvido.

Em uma conexão com um banco de dados utilizando o *framework* JDBC, é necessário realizar a instalação de um arquivo .jar na pasta do projeto, geralmente este arquivo é disponibilizado pelo próprio fabricante no site em que o banco de dados foi baixado, mas nem sempre a versão mais atual do conector necessariamente é a melhor, neste projeto houveram problemas envolvendo estes arquivos conectores.

Devido a um arquivo conector, no decorrer da análise do banco de dados PostgreSQL, a conexão era interrompida e o banco de dados desligado, de forma que não era possível reinicializa-lo por meio do console de serviços do Windows, sendo necessário reiniciar o computador para que o banco de dados ligasse de novo. Realizada a substituição deste conector por outra versão mais antiga, o sistema passou a funcionar normalmente.

A sintaxe de cada banco também criou dificuldades na hora da programação das *queries*, pequenos detalhes que existiam em um sistema, mas não em outro anulavam o comando para um *software* e era necessário adequar cada comando aos detalhes exigidos pelos gerenciadores para a correta execução da instrução.

Para que o sistema manipulador da base de dados pudesse realizar suas operações, era necessário que este obtivesse as informações das tabelas e relacionamentos contidos nos bancos de dados. A obtenção das informações era realizada por uma classe Java chamada DatabaseMetaData que importava essas informações dos sistemas gerenciadores. Essa classe não funcionou corretamente com o banco de dados PostgreSQL, tendo que sofrer uma modificação no *software* para que encontrasse o conteúdo necessário para a execução do experimento.

5.2 Trabalhos Futuros

Existem diversos seguimentos de trabalho com relação ao tema que proporcionariam não somente o entendimento técnico das ferramentas como também proporcionaria condições de melhoria de desempenho em sistemas já existentes:

- Estender a análise a um sistema multiusuário em um ambiente de concorrência proporcionaria um entendimento completo das propriedades e características de cada sistema. Para isso seria necessário aprimorar os equipamentos visto o tempo decorrido em uma máquina simples para um sistema monousuário;
- Aumentar a precisão dos resultados também necessitaria de computadores mais rápidos visto que o sistema necessitaria executar os testes mais vezes, contudo proporcionaria maior consistência das informações provenientes dos testes;
- Proporcionar diferentes métricas para o sistema também é um objetivo de um trabalho futuro, algumas delas precisam de um estudo mais aprofundado, mas produzem resultados em que é possível observar outras características dos sistemas;
- Realizar testes envolvendo diferentes configurações das variáveis internas dos sistemas gerenciadores para verificar se há um aumento significativo de desempenho contribuiria para o conhecimento do potencial que cada sistema pode atingir;
- A extensão de um sistema que gere os gráficos a partir dos valores obtidos nas análises complementaria um trabalho futuro e deixaria o sistema podendo apresentar os resultados sem a necessidade de um outro *software* que gera os gráficos, como no caso o uso do Excel neste projeto.

5.3 Trabalhos Relacionados

Existem trabalhos que estão relacionados ao estudo de benchmark entre os *softwares* abordados, sendo possível encontrar mais facilmente estudos envolvendo os bancos de dados MySQL e PostgreSQL por se tratarem de bancos de dados que

já estão a mais tempo no mercado e devido também a sua consistência de utilização pelos usuários.

Os trabalhos se diferenciam em aspectos como: base de dados utilizada, ferramentas de programação, intervalo de medidas, quantidade de iterações do algoritmo, método de processamento de dados, *hardware* utilizado, versões dos bancos de dados, entre outros fatores.

Por essa razão a comparação que pode ser realizada é a comparação do desempenho de cada operação nos bancos de dados.

O artigo escrito por Ciprian Octavian Truicã, Ion Bucur e Alexandru Boicea (2015) trata-se de uma análise comparativa realizada entre bancos de dados relacionais e não relacionais.

Essa análise foi realizada com as quantidades de operações de mil, 10 mil e 100 mil em um ambiente mono e multiusuário (OCTAVIAN, RADULESCU, *et al.*, 2015) utilizando o processamento por lotes de dados. Foram estudados bancos de dados de código aberto a partir de 50 iterações realizadas pelo algoritmo obtendo-se assim as amostras que foram analisadas.

O trabalho realizado não contém o banco de dados Firebird, por isso somente será mencionado o comportamento dos bancos de dados MySQL e PostgreSQL.

Os resultados das operações CRUD realizadas no experimento são mostrados por meio da tabela 7:

Tabela 7 – Resultados Obtidos do Trabalho Relacionado

Operation	Databank	Number of Operations		
		1000	10000	100000
Insert	MySQL	757,1	7326,4	76725,7
	PostgreSQL	80,9	798,7	10476,7
Select	MySQL	4,1	117,8	844,8
	PostgreSQL	3,7	19,4	663,5
Update	MySQL	87,7	1264	10620,5
	PostgreSQL	77,3	2385,2	25421,5
Delete	MySQL	78,3	825,8	18794,4
	PostgreSQL	35,5	582,6	11479,8

Fonte: (OCTAVIAN, RADULESCU, *et al.*, 2015)

Algumas observações podem ser feitas por meio dos resultados obtidos pelo estudo: primeiramente a dificuldade que o MySQL tem de inserir informações na base de dados frente ao PostgreSQL, a diferença para a última quantidade de operações inseridas foi de 76,72s para o primeiro contra 10,47s do segundo, mas ao longo do experimento já é possível observar a diferença de desempenho entre um sistema e outro logo na primeira quantidade de operações apresentada, nesta o MySQL inseriu mil informações em 757,1ms e o PostgreSQL em 80,9ms, logo o PostgreSQL já apresentou uma velocidade maior na execução frente ao MySQL em 9,35 vezes.

Outra observação que é possível retirar deste trabalho é o crescimento do tempo frente ao volume de dados que o PostgreSQL apresentou na operação de seleção de dados. Apesar do PostgreSQL ter finalizado o ciclo de operações de forma mais rápida que o MySQL nesta operação, é possível observar, da primeira quantidade de operações apresentada para 10 mil, que o PostgreSQL iniciou o ciclo de testes em 3,7ms e obteve 19,4ms como o resultado da quantidade de 10 mil operações, caracterizando um aumento no tempo de 15,7ms, em contrapartida, o MySQL iniciou os testes em 4,1ms e chegou em 117,8ms para 10 mil operações, obtendo assim uma diferença de 113,7ms.

Conforme ocorreu o aumento da quantidade de operações, de 10 mil para 100 mil, o PostgreSQL aumentou o tempo de execução do lote de operações assim como o MySQL, visto que houve o aumento de informação que foi selecionada atrelado a dificuldade de selecionar os valores com um volume de dados maior. Neste intervalo de valores, o PostgreSQL elevou o tempo de execução de 19,4ms com 10mil operações para 663,5ms com 100mil operações, caracterizando uma diferença de 644,1ms, em contrapartida, o MySQL executou 10 mil operações em 117,8ms e 100 mil em 844,8ms, com a diferença de tempo de 727ms.

Visto o aumento do tempo em que o PostgreSQL executou as operações conforme a variação do volume de dados, pode-se dizer que conforme o aumento do mesmo o MySQL poderá passar a executar as operações de forma mais rápida que o PostgreSQL, pois dividindo o tempo do último lote de operações pelo antepenúltimo, o resultado mostra que houve o aumento de 7,17 vezes entre o tempo das quantidades de operações de 10 mil e 100 mil para o MySQL, enquanto que para o PostgreSQL esse aumento foi de 34,2 vezes.

O software de benchmark apresentado neste trabalho forneceu medidas que apresentaram o aumento do tempo do PostgreSQL frente a grandes volumes de

dados, mostrando ao final do ciclo de operações diferenças de desempenho entre esses bancos de dados para as operações de seleção.

As operações de modificação entre os dois bancos de dados apresentada no artigo mantiveram tempos semelhantes na primeira quantidade de operações, 77,3ms para o PostgreSQL frente a 87,7ms do MySQL. Conforme ocorreu o aumento da quantidade de operações, o PostgreSQL obteve piores resultados que o MySQL com diferenças de tempo entre os dois bancos de dados de 1,12s com 10 mil operações e 14,8s com 100 mil.

Para as operações de exclusão de dados, o experimento realizado no artigo foi semelhante ao da inserção de valores, o PostgreSQL desde a primeira quantidade de operações executou-as mais rapidamente que o MySQL, contudo a diferença de desempenho entre esses bancos de dados para esta operação não foi tão grande como na operação de inserção, pois no início do ciclo de testes a diferença de tempo entre os bancos de dados para as operações de exclusão foi de 42,8ms frente a 676,2ms das operações de inserção, seguindo também com grandes diferenças de desempenho para as quantidades de operações subsequentes.

O *software* de *benchmark* construído neste projeto conseguiu apresentar resultados que identificaram as diferenças de desempenho entre os bancos de dados MySQL e PostgreSQL para as operações assim como os resultados descritos no artigo. Nas operações de inserção e exclusão apresentadas neste projeto, o *software* apontou diferenças de desempenho que favoreceram o PostgreSQL, contudo não foi possível identifica-las somente com os volumes de dados apresentados no trabalho, logo houve a necessidade de realizar os testes com volumes de dados maiores.

Outro ponto que deve ser destacado na comparação entre os resultados fornecidos pelo *software* de *benchmark* construído neste projeto e os fornecidos pelo artigo é a respeito das operações de maior custo, pois apesar dos valores dos intervalos de medida serem diferentes é possível visualizar que tanto os resultados obtidos pelo software construído quanto os obtidos pelo artigo mostram que a operação de maior custo para o MySQL é a inserção de valores e para o PostgreSQL é a modificação de valores.

Os estudos realizados mostram que apesar de se tratarem de bancos de dados que seguem o mesmo paradigma de armazenamento de dados e a mesma linguagem de comunicação, estes possuem processamentos diferentes para cada operação que produzem desempenhos melhores e piores quando analisados com outros bancos.

6.0 REFERÊNCIAS

ALVARO, F. **Easy SQL Programming and Database Management For Beginners. Your Step-by-Step Guide To Learning the SQL Database.** 1. ed. [S.l.]: [s.n.], 2016.

ANDERSON, K. 2019 Open Source Database Report. **DZone**, 2019. Disponivel em: <https://dzone.com/articles/2019-open-source-database-report-top-databases-pub>.

BONNICI, T. S. **Benchmarking**. [S.l.]: [s.n.], 2015. 3 p. Disponivel em: <https://www.researchgate.net/publication/272352934_Benchmarking>. Acesso em: 2020 jan. 05.

BRILL, G. **CodeNotes for J2EE:** EJB, JDBC, JSP and Servlets. [S.l.]: Random House Trade Paperbacks, 2002. 243 p.

BRITO, R. W. **Bancos de Dados Nosql x SGBDs relacionais: Análise comparativa**, 2010. Disponivel em: <http://www.infobrasil.inf.br/userfiles/27-05-S4-1-68840-Bancos%20de%>.

BUTTERWORTH, H. **The Benchmarking Book:** A How-to-Guide to Best Practice for Managers and Practitioners. [S.l.]: Elsevier Ltd, 2009.

CANTU, C. H. **Get to know Firebird in 2 minutes**, 2010. Disponivel em: <https://www.firebirdnews.org/docs/fb2min.pdf>.

DARMONT, J. **Database Benchmarks**, Janeiro 2017. 18.

DATE, C. J. **An Introduction to Database**. [S.l.]: Pearson; 8 edition, 2004.

DEWITT, D. J. **The Wisconsin Benchmark: Past, Present, and Future**, 1991. 43.

DIN, A. L. **Structured Query Language (SQL) A Practical Introduction**. [S.l.]: BLACKWELL, 2012.

ELMASRI, R. **Fundamentals of data base systems**. [S.l.]: EOMBD UNIVERSITY, 1994.

FARIA, T.; JUNIOR, N. **JPA e Hibernate**. 1ª. ed. [S.l.]: AlgaWorks, 2015.

FERRARI; DOMENICO. **Computer System Performance Evaluation**. [S.l.]: Prentice Hall, 1978.

FIREBIRD, 05 fev. 2020. Disponivel em: <https://firebirdsql.org/en/firebird-rdbms/>.

GÖETZ, B. et al. **Java Concurrency In Practice**. [S.l.]: Addison-Wesley Professional, 2006.

GRAY, J. **Database and Transaction Processing Performance Handbook**. [S.l.]: [s.n.], 1993.

HALVORSEN, H. P. **Structured Query Language**. [S.l.]: University College of Southeast Norway, 2016.

IVANOV1, T. et al. **Big Data Benchmark Compendium**. 20. Disponivel em: <http://people.ac.upc.edu/npoggi/publications/SPEC-RGBD-TPCTC-2015-Big%20Data%20Benchmark%20Compendium.pdf>.

JAIN, R. **The Art of Computer Systems Performance Analysis**. [S.l.]: Wiley, 1991.

LASCANO, J. E. JPA implementations versus pure JDBC, Sangolquí, Junho 2008. 11. Disponivel em: <https://www.researchgate.net/publication/237449456_JPA_implementations_versus_pure_JDBC>.

LUCIDCHART. Entity-Relationship Diagram Symbols and Notation. **LucidChart**, 02 abr. 2020. Disponivel em: <https://www.lucidchart.com/pages/ER-diagram-symbols-and-meaning>.

MACÁRIO, C. G. D. N.; BALDO, S. M. O Modelo Relacional, Campinas. 15.
MARTIN, B. **JDBC Developer's Guide and Reference**. [S.l.]: [s.n.], 2010.

MOLINA, H. G.; ULMAN, J. D.; WIDOM, J. **Database System:** The Complete Book. [S.l.]: Prentice Hall, 1998.

OAKS, S.; WONG, H. **Java Threads**. [S.l.]: O'Reilly, 1999.

OCTAVIAN, C. et al. Performance evaluation for CRUD operations in asynchronously replicated document oriented database, Bucharest, Maio 2015. 7. Disponivel em: <https://www.researchgate.net/publication/280832003_Performance_Evaluation_for_CRUD_Operations_in_Asynchronously_Replicated_Document_Oriented_Database>. Acesso em: 2020 mar. 10.

OLIVEIRA, H. E. M. JPA Passo a Passo. **The Developer´s Conference**, 2008. Disponivel em: <https://thedevconf.com/arquivos/TDC2008Floripa-jpa-henrique.pdf>. Acesso em: 2020 mar. 02.

RAMAKRISHNAN, R.; GEHRKE, J. **Data Base Management Systens**. [S.l.]: [s.n.], 2009.

RUSSELL, J. **Orable MySQL Essentials**. [S.l.]: [s.n.], 2015.

SADOGHI, M.; BHATTACHERJEE, S.; CANIM, M. L-Store: A Real-time OLTP and OLAP System, abril 2018. 13. Disponivel em: <https://www.researchgate.net/publication/291229576_L-Store_A_Real-time_OLTP_and_OLAP_System>.

SCALZO, B. **Data Base Benchmarking and Stress Testing**. Flower Mound: [s.n.], 2018.

SENG, J.-L. A study on industry and synthetic standard benchmarks in relational and object databases, 20 Maio 2015. 18.

SMYTH, N. **MySQL Essentials**. [S.l.]: Payload Media, 2007.

SPEEGLE, G. D. **JDBC:** Practical Guide for Java Programmers. 1ª. ed. [S.l.]: Morgan Kaufmann, 2001.

STAIR, R.; REYNOLDS, G. **Principles of Information Systems**. [S.l.]: Cengage Learning, 2017.

STONEBRAKER, M.; ROWE, L. A. **THE DESIGN OF POSTGRES**. Berkeley: University of California, 1986.

STRAWSER, P. R. **A Methodology For Benchmarking Relational Database Machine**. California: [s.n.], 1984.

TAKAI, O.; ITALIANO, I.; FERREIRA, J. Introdução a banco de dados, 2005. Disponivel em: <http://dsc.inf.furb.br/arquivos/tccs/monografias/TCC2012-2-22-VF-PauloABugmann.pdf>.

THE Main Features of MySQL. **MySQL**, 09 jan. 2020. Disponivel em: <https://dev.mysql.com/doc/refman/8.0/en/features.html>.

THE Postgre Global Development Group. **Documentação do PostgreSQL 9.3**, 08 jan. 2020. Disponivel em: <http://www.postgresql.org/docs/9.3/interactive/index.html>.

VENNERS, B. **Inside the Java Virtual Machine**. [S.l.]: McGraw-Hill Osborne Media, 1998.

APÊNDICES

A – Classe Principal

```
package apptcc;

import static java.lang.Integer.parseInt;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.util.Pair;
import javax.swing.JTextField;

/**
 *
 * @author Rafhael
 */
public class JFrameBENCHMARK extends javax.swing.JFrame {

    /**
     * Creates new form JFrameBENCHMARK
     */
    private static int INSERT;
    private static int SELECT;
    private static int UPDATE;
    private static int DELETE;
    private static int INCREMENT_INSERT;
    private static int INCREMENT_SELECT;
    private static int INCREMENT_UPDATE;
    private static int INCREMENT_DELETE;
    private static String FILE_NAME;
    private static int ROW;
    private static int COW;
    private static int AVERAGE;
    private static boolean SHOW_PRINT_MESSAGES;
    private static boolean BATCH_PROCESSING;
    private static int TABLE_NUMBER;
    private static List<Integer> OPTION;
    private static int BANCKS;
    private static List<List<Object>> PRESETS;
    private static int presetCounter;
    private static boolean READY;

    private static Execution ex;

    public JFrameBENCHMARK() {
        initComponents();

        //__Inicial program definitions_____________________________________//
        BATCH_PROCESSING = true;
        SHOW_PRINT_MESSAGES = false;
        ROW = 25;
        COW = 4;
        READY = false;
        jTextField1.setEnabled(false);
        jTextField2.setEnabled(false);
        jTextField1.setText("0");
        jTextField2.setText("0");
        jTextField3.setText("1");
        jTextField4.setText("1");
        jCheckBox1.setEnabled(false);
        //                                                                  //
        //__________________________________________________________________//
        //__Form definitions________________________________________________//
        jTextArea1.setEditable(false);
        //                                                                  //
        //__________________________________________________________________//
        //--Define PRESETS to put in text fields----------------------------//
        presetCounter = 0;
        List<Object> presets = new ArrayList<>();
        PRESETS = new ArrayList<>();

        //__Preset 1________________________________________________________//
        presets.add("500");
        presets.add("50");
        presets.add("250");
        presets.add("1");
        PRESETS.add(presets);
        presets = new ArrayList<>();
        //                                                                  //
        //__________________________________________________________________//
        //__Preset 2________________________________________________________//
        presets.add("500000");
        presets.add("50000");
        presets.add("50");
        presets.add("1");
        PRESETS.add(presets);
        presets = new ArrayList<>();
        //__________________________________________________________________//
        //__Preset 3________________________________________________________//
```

```
        presets.add("10000");
        presets.add("1000");
        presets.add("1");
        presets.add("1");
        PRESETS.add(presets);
        presets = new ArrayList<>();
        //-------------------------------------------------------------------//
        //-------------------------------------------------------------------//
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jPanel1 = new javax.swing.JPanel();
        jScrollPane1 = new javax.swing.JScrollPane();
        jTextArea1 = new javax.swing.JTextArea();
        jTextField1 = new javax.swing.JTextField();
        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jTextField2 = new javax.swing.JTextField();
        jLabel3 = new javax.swing.JLabel();
        jTextField3 = new javax.swing.JTextField();
        jTextField4 = new javax.swing.JTextField();
        jLabel5 = new  javax.swing.JLabel();
        jLabel6 = new javax.swing.JLabel();
        jCheckBox2 = new javax.swing.JCheckBox();
        jButton1 = new javax.swing.JButton();
        jComboBox1 = new javax.swing.JComboBox<>();
        jLabel7 = new  javax.swing.JLabel();
        jLabel8 = new javax.swing.JLabel();
        jTextField5 = new javax.swing.JTextField();
        jButton2 = new javax.swing.JButton();
        jButton4 = new javax.swing.JButton();
        jLabel4 = new javax.swing.JLabel();
        jCheckBox1 = new javax.swing.JCheckBox();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("BENCHMARK SOFTWARE");

        jPanel1.setBackground(new java.awt.Color(255, 255, 255));


        jTextArea1.setBackground(new java.awt.Color(0, 0, 0));
        jTextArea1.setColumns(20);
        jTextArea1.setForeground(new java.awt.Color(255, 255, 255));
        jTextArea1.setRows(5);
        jScrollPane1.setViewportView(jTextArea1);

        jTextField1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jTextField1ActionPerformed(evt);
            }
        });

        jLabel1.setText("Operations:");

        jLabel2.setText("Increments:");

        jTextField2.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jTextField2ActionPerformed(evt);
            }
        });

        jLabel3.setText("Average:");

        jTextField3.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jTextField3ActionPerformed(evt);
            }
        });

        jTextField4.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jTextField4ActionPerformed(evt);
            }
        });

        jLabel5.setText("Table Number:");

        jLabel6.setText("Option Mode SQL Test:");

        jCheckBox2.setText("(DML and DQL)/ (DDL)");
        jCheckBox2.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jCheckBox2ActionPerformed(evt);
            }
```

```
        });

        jButton1.setText("Start");
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });

        jComboBox1.setModel(new javax.swing.DefaultComboBoxModel<>(new String[] { "MySQL", "PostgreSQL", "Firebird", "MySQL and PostgreSQL", "MySQL an
        jComboBox1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jComboBox1ActionPerformed(evt);
            }
        });

        jLabel7.setText("Data Bank:");

        jLabel8.setText("File Path");

        jTextField5.setText("C:/Users/Rafhael/Desktop/TCC/Resultados/Resultados");
        jTextField5.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jTextField5ActionPerformed(evt);
            }
        });

        jButton2.setText("Presets");
        jButton2.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton2ActionPerformed(evt);
            }
        });

        jButton4.setText("Clear");
        jButton4.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton4ActionPerformed(evt);
            }
        });

        jLabel4.setText("Show Console Messages:");

        jCheckBox1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jCheckBox1ActionPerformed(evt);

            }
        });

        javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
        jPanel1.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 695, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, jPanel1Layout.createSequentialGroup()
                        .addGap(0, 22, Short.MAX_VALUE)
                        .addComponent(jButton4)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(jButton2)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(jButton1, javax.swing.GroupLayout.PREFERRED_SIZE, 65, javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addComponent(jTextField1)
                    .addComponent(jTextField2)
                    .addComponent(jTextField3)
                    .addComponent(jTextField4)
                    .addComponent(jTextField5, javax.swing.GroupLayout.PREFERRED_SIZE, 0, Short.MAX_VALUE)
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addComponent(jLabel7)
                        .addContainerGap())
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                            .addComponent(jComboBox1, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupL
                            .addComponent(jLabel1)
                            .addComponent(jLabel2)
                            .addComponent(jLabel3)
                            .addComponent(jLabel5)
                            .addComponent(jLabel8)
                            .addComponent(jLabel4)
                            .addComponent(jLabel6)
                            .addComponent(jCheckBox2)
                            .addComponent(jCheckBox1))
                        .addGap(0, 0, Short.MAX_VALUE))))
        );
        jPanel1Layout.setVerticalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jScrollPane1)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jLabel8
```

```
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jTextField5, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFE
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jLabel6)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jCheckBox2)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jLabel1)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jTextField1, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFE
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jLabel2)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jTextField2, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFE
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jLabel3)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jTextField3, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFE
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jLabel5)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jTextField4, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFE
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jLabel4)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jCheckBox1)
            .addGap(8, 8, 8)
            .addComponent(jLabel7)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jComboBox1, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFER
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(jButton4)
                .addComponent(jButton2)
                .addComponent(jButton1))
            .addGap(0, 0, Short.MAX_VALUE))
    );

    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
    getContentPane().setLayout(layout);
    layout.setHorizontalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
    );
    layout.setVerticalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
    );

    pack();
}// </editor-fold>

private void jCheckBox2ActionPerformed(java.awt.event.ActionEvent evt) {
    if (jCheckBox2.isSelected()) {
        OPTION = new ArrayList<>(Arrays.asList(1, 2, 4, 5));
        jTextField1.setEnabled(true);
        jTextField2.setEnabled(true);
        jCheckBox1.setEnabled(true);
    } else {
        OPTION = new ArrayList<>(Arrays.asList(3));
        jTextField1.setEnabled(false);
        jTextField2.setEnabled(false);
        jTextField1.setText("0");
        jTextField2.setText("0");
        jCheckBox1.setEnabled(false);
    }
}

private void jTextField1ActionPerformed(java.awt.event.ActionEvent evt) {
    INSERT = parseInt(jTextField1.getText());
    SELECT = parseInt(jTextField1.getText());
    UPDATE = parseInt(jTextField1.getText());
    DELETE = parseInt(jTextField1.getText());
}

private void jTextField2ActionPerformed(java.awt.event.ActionEvent evt) {
    INCREMENT_INSERT = parseInt(jTextField2.getText());
    INCREMENT_SELECT = parseInt(jTextField2.getText());
    INCREMENT_UPDATE = parseInt(jTextField2.getText());
    INCREMENT_DELETE = parseInt(jTextField2.getText());
}

private void jTextField3ActionPerformed(java.awt.event.ActionEvent evt) {
    AVERAGE = parseInt(jTextField3.getText());
}

private void jTextField4ActionPerformed(java.awt.event.ActionEvent evt) {
    TABLE_NUMBER = parseInt(jTextField4.getText());
}

private void jComboBox1ActionPerformed(java.awt.event.ActionEvent evt) {
    BANCKS = jComboBox1.getSelectedIndex();
```

```
}

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    jTextField5ActionPerformed(evt);
    jTextField1ActionPerformed(evt);
    jTextField2ActionPerformed(evt);
    jTextField3ActionPerformed(evt);
    jCheckBox2ActionPerformed(evt);
    jTextField4ActionPerformed(evt);
    jComboBox1ActionPerformed(evt);
    Execution ex = new Execution();
    ex.start();
}

private void jTextField5ActionPerformed(java.awt.event.ActionEvent evt) {
    FILE_NAME = jTextField5.getText();
}

private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    if(jTextField1.isEnabled()){
        jTextField1.setText((String) PRESETS.get(presetCounter).get(0));
    }
    if(jTextField2.isEnabled()){
        jTextField2.setText((String) PRESETS.get(presetCounter).get(1));
    }
    jTextField3.setText((String) PRESETS.get(presetCounter).get(2));
    jTextField4.setText((String) PRESETS.get(presetCounter).get(3));
    if (presetCounter < PRESETS.size() - 1) {
        presetCounter++;
    } else {
        presetCounter = 0;
    }
}

private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
    jTextArea1.setText("");
}

private void jCheckBox1ActionPerformed(java.awt.event.ActionEvent evt) {
    if (jCheckBox1.isSelected()) {
        SHOW_PRINT_MESSAGES = true;
    } else {
        SHOW_PRINT_MESSAGES = false;
    }
}

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new JFrameBENCHMARK().setVisible(true);
        }
    });
}

// Variables declaration - do not modify
private javax.swing.JButton jButton1;
private javax.swing.JButton jButton2;
private javax.swing.JButton jButton4;
private javax.swing.JCheckBox jCheckBox1;
private javax.swing.JCheckBox jCheckBox2;
private javax.swing.JComboBox<String> jComboBox1;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel5;
private javax.swing.JLabel jLabel6;
private javax.swing.JLabel jLabel7;
private javax.swing.JLabel jLabel8;
private javax.swing.JPanel jPanel1;
private javax.swing.JScrollPane jScrollPane1;
private javax.swing.JTextArea jTextArea1;
private javax.swing.JTextField jTextField1;
private javax.swing.JTextField jTextField2;
private javax.swing.JTextField jTextField3;
private javax.swing.JTextField jTextField4;
private javax.swing.JTextField jTextField5;
// End of variables declaration

class Execution extends Thread {

    //__Variables execute program_____________________________________//
    List<Double> data = new ArrayList<>();
    List<List<Object>> printMessages = new ArrayList<>();
    WriterExcel wre = new WriterExcel(); //write in Excel
    int avg = AVERAGE;
    int auxAvg = 0;
    List<String> dbNames = new ArrayList<>();
```

```
Object[][] printExcelCRUD = new Object[ROW][COW];
Object[][] printExcelCreateDrop = new Object[ROW][COW];
List<Integer> dbs = new ArrayList<>();
//…… //

public void run() {

    //…… //
    if (!jCheckBox2.isSelected()) {
        jTextArea1.append("_____" + "\n");
        jTextArea1.append("Analysing DDL language" + "\n");
        jTextArea1.append("_____" + "\n");
        testTablesDataBase(true);
    } else {
        testTablesDataBase(false);
    }
    //…… //

    if (!jCheckBox2.isSelected()) {
        //--Print results of functions database function in excel file --- //
        jTextArea1.append("##########################################################################" + "\n");
        jTextArea1.append("Creating excel file" + "\n");
        wre.write(printExcelCreateDrop, FILE_NAME + "SemAutoCommit/DDLTest_Tables("+ TABLE_NUMBER + ")" + "_Average"+"("+AVERAGE+")"+"_Option(
        printExcelCreateDrop = new Object[ROW][COW];
        jTextArea1.append("Done" + "\n");
        jTextArea1.append("##########################################################################" + "\n");
        //…… //
    }

    if (jCheckBox2.isSelected()) {
        //…… //
        int vIns = 0;
        int vSel = 0;
        int vUp = 0;
        int vDel = 0;
        //…… //

        //…… //
        double insertTime = 0;
        double selectTime = 0;
        double updateTime = 0;
        double deleteTime = 0;
        //…… //

        //…… //
        int row = 0;
        int cow = 0;
        //…… //
        for (int i = 0; i < dbs.size(); i++) {
            row = 0;
            int countAnalysing = 1;
            while (vIns < INSERT) {

                auxAvg = avg;
                vSel += INCREMENT_SELECT;
                vUp += INCREMENT_UPDATE;
                vIns += INCREMENT_INSERT;
                vDel += INCREMENT_DELETE;

                jTextArea1.append("**********************************************************************" + "\n");
                jTextArea1.append("(" + dbNames.get(i) + ") " + "analysing " + countAnalysing + " iteration..." + "\n");
                jTextArea1.append("**********************************************************************" + "\n");
                jTextArea1.setCaretPosition(jTextArea1.getDocument().getLength());

                while (auxAvg > 0) {
                    Pair<List<Double>, List<List<Object>>> dataAndPrintLogs = null;
                    try {
                        dataAndPrintLogs = DataBaseAutoM.executeManipulator(vIns, vSel, vUp, vDel, dbs.get(i), OPTION, SHOW_PRINT_MESSAGES, BA
                    } catch (SQLException ex) {
                        Logger.getLogger(JFrameBENCHMARK.class.getName()).log(Level.SEVERE, null, ex);
                    }
                    data = dataAndPrintLogs.getKey();
                    printMessages = dataAndPrintLogs.getValue();
                    if (SHOW_PRINT_MESSAGES == true) {
                        for (int messageLogsList = 0; messageLogsList < printMessages.size(); messageLogsList++) {
                            jTextArea1.append("_____" + "\n");
                            for (int messageLogs = 0; messageLogs < printMessages.get(messageLogsList).size(); messageLogs++) {
                                jTextArea1.append((String) printMessages.get(messageLogsList).get(messageLogs) + "\n");
                                jTextArea1.setCaretPosition(jTextArea1.getDocument().getLength());
                            }
                            jTextArea1.append("_____" + "\n");
                        }
                    }
                    insertTime += data.get(0);
                    selectTime += data.get(1);
                    updateTime += data.get(2);
                    deleteTime += data.get(3);
                    auxAvg--;
                }
                if (insertTime > 0) {
                    insertTime = (double) insertTime / avg;
                    printExcelCRUD[row][cow] = insertTime;
```

```
                    cow++;
                }
                if (selectTime > 0) {
                    selectTime = (double) selectTime / avg;
                    printExcelCRUD[row][cow] = selectTime;
                    cow++;
                }
                if (updateTime > 0) {
                    updateTime = (double) updateTime / avg;
                    printExcelCRUD[row][cow] = updateTime;
                    cow++;
                }
                if (deleteTime > 0) {
                    deleteTime = (double) deleteTime / avg;
                    printExcelCRUD[row][cow] = deleteTime;
                }
                cow =  0;
                row++;
                insertTime = 0;
                selectTime = 0;
                updateTime = 0;
                deleteTime = 0;
                countAnalysing++;
            }
            vSel = 0;
            vUp = 0;
            vIns = 0;
            vDel = 0;

            jTextArea1.append("#############################################################################" + "\n");
            jTextArea1.append("Creating excel file" + "\n");
            wre.write(printExcelCRUD, FILE_NAME + "SemAutoCommit/" + dbNames.get(i) + "SAC_VDADOS(" + INSERT + ")_INC(" + INCREMENT_INSERT + "
            printExcelCRUD = new Object[ROW][COW];
            jTextArea1.append("Done" + "\n");
            jTextArea1.append("#############################################################################" + "\n");

        }
    }
}

public void testTablesDataBase(boolean DDLLanguage) {
    List<Pair<Double, Double>> resultCreateDrop = new ArrayList<>();
    List<Pair<Double, Double>> avgCreateDrop = new ArrayList<>();
    double dropTableRes = 0;
    double createTableRes = 0;
    double dropTableAVG = 0;
    double createTableAVG = 0;
    int key1 = 0;
    if (DDLLanguage == true) {
        for (int i = 1; i <= AVERAGE; i++) {
            resultCreateDrop = switchTestTablesDataBase();
            for (int j = 0; j < resultCreateDrop.size(); j++) {
                if(key1 == 0){
                    dropTableAVG = resultCreateDrop.get(j).getKey();
                    createTableAVG = resultCreateDrop.get(j).getValue();
                    avgCreateDrop.add(new Pair(dropTableAVG, createTableAVG));
                }else{
                    dropTableRes = resultCreateDrop.get(j).getKey();
                    createTableRes = resultCreateDrop.get(j).getValue();
                    dropTableAVG = avgCreateDrop.get(j).getKey() + dropTableRes;
                    createTableAVG = avgCreateDrop.get(j).getValue() + createTableRes;
                    avgCreateDrop.set(j, new Pair(dropTableAVG,createTableAVG));
                }
            }
            key1 = 1;
        }
        for(int i = 0; i < avgCreateDrop.size(); i++){
            dropTableAVG = (double)avgCreateDrop.get(i).getKey()/AVERAGE;
            createTableAVG = (double)avgCreateDrop.get(i).getValue()/AVERAGE;
            avgCreateDrop.set(i, new Pair(dropTableAVG,createTableAVG));
        }
        int counter = 0;
        for (int i = 0; i < avgCreateDrop.size(); i++) {
            printExcelCreateDrop[i][0] = avgCreateDrop.get(i).getKey();
            printExcelCreateDrop[i][1] = avgCreateDrop.get(i).getValue();
        }
    } else {
        switchTestTablesDataBase();
    }
}

public List<Pair<Double, Double>> switchTestTablesDataBase() {
    List<Pair<Double, Double>> listResultCreateDrop = new ArrayList<>();
    Pair<Double, Double> resultCreateDrop = null;
    switch (BANCKS) {
        case 0:
            dbs.add(0);
            resultCreateDrop = DataBaseAutoG.executeGenerator(dbs.get(0), TABLE_NUMBER, null);
            listResultCreateDrop.add(resultCreateDrop);
            dbNames.add("MySQL");
            break;
        case 1:
```

```
                dbs.add(1);
                resultCreateDrop = DataBaseAutoG.executeGenerator(dbs.get(0), TABLE_NUMBER, null);
                listResultCreateDrop.add(resultCreateDrop);
                dbNames.add("PostgreSQL");
                break;
            case 2:
                dbs.add(2);
                resultCreateDrop = DataBaseAutoG.executeGenerator(dbs.get(0), TABLE_NUMBER, null);
                dbNames.add("Firebird");
                break;
            case 3:
                dbs.add(0);
                dbs.add(1);
                resultCreateDrop = DataBaseAutoG.executeGenerator(dbs.get(0), TABLE_NUMBER, null);
                listResultCreateDrop.add(resultCreateDrop);
                resultCreateDrop = DataBaseAutoG.executeGenerator(dbs.get(1), TABLE_NUMBER, null);
                listResultCreateDrop.add(resultCreateDrop);
                dbNames.add("MySQL");
                dbNames.add("PostgreSQL");
                break;
            case 4:
                dbs.add(0);
                dbs.add(2);
                resultCreateDrop = DataBaseAutoG.executeGenerator(dbs.get(0), TABLE_NUMBER, null);
                listResultCreateDrop.add(resultCreateDrop);
                resultCreateDrop = DataBaseAutoG.executeGenerator(dbs.get(1), TABLE_NUMBER, null);
                listResultCreateDrop.add(resultCreateDrop);
                dbNames.add("MySQL");
                dbNames.add("Firebird");
                break;
            case 5:
                dbs.add(1);
                dbs.add(2);
                resultCreateDrop = DataBaseAutoG.executeGenerator(dbs.get(0), TABLE_NUMBER, null);
                listResultCreateDrop.add(resultCreateDrop);
                resultCreateDrop = DataBaseAutoG.executeGenerator(dbs.get(1), TABLE_NUMBER, null);
                listResultCreateDrop.add(resultCreateDrop);
                dbNames.add("PostgreSQL");
                dbNames.add("Firebird");
                break;
            case 6:
                dbs.add(0);
                dbs.add(1);
                dbs.add(2);
                resultCreateDrop = DataBaseAutoG.executeGenerator(dbs.get(0), TABLE_NUMBER, null);
                listResultCreateDrop.add(resultCreateDrop);
                resultCreateDrop = DataBaseAutoG.executeGenerator(dbs.get(1), TABLE_NUMBER, null);
                listResultCreateDrop.add(resultCreateDrop);
                resultCreateDrop = DataBaseAutoG.executeGenerator(dbs.get(2), TABLE_NUMBER, null);
                listResultCreateDrop.add(resultCreateDrop);
                dbNames.add("MySQL");
                dbNames.add("PostgreSQL");
                dbNames.add("Firebird");
        }
        return listResultCreateDrop;
    }
}
```

B – Gerador de Base de Dados

```
package apptcc;

import javafx.util.Pair;
import java.sql.Connection;
import java.sql.DatabaseMetaData;
import java.sql.DriverManager;
import  java.sql.ResultSet;
import java.sql.SQLException;
import  java.sql.Statement;
import  java.time.Duration;
import java.time.Instant;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.Scanner;

/**
 *
 * @author Rafhael
 */
public class DataBaseAutoG {

    //__Uso Geral_____________________________________________________________//
    public static Scanner scanner = new Scanner(System.in);
    public static Random rand = new Random(6754L);     //7173246411312
    public static DBConnection dbConnection = new DBConnection();
    //________________________________________________________________________//

    //__Variables List________________________________________________________//
    public static String varchar = "VARCHAR";
    public static String integer = "INT";
    public static String sfloat = "FLOAT";
    public static String sdate = "DATE";
    //________________________________________________________________________//

    public static Pair<Double, Double> executeGenerator(int database, int tNumber, List<Pair<String, Pair<String, String>>> listR) {

        //--Variables to create and drop database-----------------------------//
        List<String> tables = new ArrayList<>();
        int tableNumber = tNumber;
        List<Pair<String, Pair<String, String>>> listRelations = listR;

        List<String> listForeignKeys = new ArrayList<>();
        //                                                                    //
        //--Remove constraints and tables in a database----------------------//
        double dropTime = dropLoop(database, tables, listForeignKeys);
        //……….……//
        //__Create Tables_________________________________________________//
        double createTime = createLoop(tableNumber, database);
        //……….……//
        //__Create E/R Model______________________________________________//
        //System.out.println("Creating E/R Model");
        //createERModel(database, listRelations);
        //                                                                    //

        return new Pair(dropTime, createTime);
    }

    //**Create Table Functions*************************************************//
    //__Just to continue Execute Function_____________________________________//
    public static double createLoop(int tableNumber, int database) {
        int randVariables = 0;
        int randType = 0;
        List<Pair<Integer, List<Integer>>> randsCreation = new ArrayList<>();
        double createTime = 0;
        //--Stablish all variables and types in the tables--------------------//
        for (int i = 0; i < tableNumber; i++) {
            randVariables = 6;//rand.nextInt(5) + 1;
            List<Integer> randTypeList = new ArrayList<>();
            for (int j = 0; j < randVariables; j++) {
                randType = rand.nextInt(4);
                randTypeList.add(randType);
            }
            randsCreation.add(new Pair(randVariables, randTypeList));
        }
        //                                                                    //
        try (Connection connection = DriverManager.getConnection(dbConnection.getUrl().get(database), dbConnection.getUser().get(database), dbConnecti
          ……Statement statement = connection.createStatement();……
                long startCreate = System.nanoTime();
                createTableDataBase(connection, randsCreation, tableNumber);
                long finishCreate = System.nanoTime() - startCreate;
                createTime = (double) finishCreate / 1000000;
              for (int j = 0; j < tableNumber; j++) {
                changeAutoIncFromTable(connection, database, j);
            }

            connection.close();
        } catch (SQLException e) {
```

```
            throw new IllegalStateException("DataBaseAutoGenerator1.Main:error", e);
        }
        return createTime;
    }

    //……//
    //__Create a database with tables_____//
    public static void createTableDataBase(Connection connection, List<Pair<Integer, List<Integer>>> randsCreation, int tables) throws SQLException {
        connection.setAutoCommit(false);
        Statement statement = connection.createStatement();
        for (int i = 0; i < tables; i++) {
            String query = "CREATE TABLE T" + i + " (";
            int randVariables = randsCreation.get(i).getKey();
            //--Choose variables according to the list of variables
            for (int j = 0; j < randVariables; j++) {
                int randType = randsCreation.get(i).getValue().get(j);
                switch (randType) {
                    case 0:
                        query = query + "vName" + j + " " + varchar + "(1000)" + ",";
                        break;
                    case 1:
                        query = query + "vName" + j + " " + integer + ",";
                        break;
                    case 2:
                        query = query + "vName" + j + " " + sfloat + ",";
                        break;
                    case 3:
                        query = query + "vName" + j + " " + sdate + ",";
                }
            }
            StringBuilder sbuilder = new StringBuilder(query);
            sbuilder.setCharAt(query.length() - 1, ')');
            query = sbuilder.toString();
            statement.addBatch(query);
        }
        statement.executeBatch();
        connection.commit();
    }
    //……//
    //**********************************************************************//

    //--Create relation 1x1 cbetween tables in the data base------------------//
    //__oo is one to one_____//
    static void createRelation1x1(Connection connection, String tName, String tRefName, String tIndex, String tRefIndex) {
        try {
            String query = "ALTER TABLE " + tName + " ADD CONSTRAINT " + "fkoo" + tName + tRefName + " FOREIGN KEY (" + tIndex + ")" + " REFERENCES "

            Statement st = connection.createStatement();
            st.executeUpdate(query);
        } catch (SQLException e) {
            throw new IllegalStateException("DataBaseAutoGenerator1.createRelation1x1: Error", e);
        }
    }
    //……//

    //--Create relation 1xn cbetween tables in the data base------------------//
    //__om: one to many_____//
    //--fkom: foreign key relation one to many------------------------------//
    static void createRelation1xN(Connection connection, String tName, String tRefName, String tRefIndex) {
        try {
            Statement st = connection.createStatement();
            String refTableVariable = "fkom" + tRefName;
            String fCon = connection.toString();
            String query = "";
            if (fCon.contains("firebird")) {
                query = "ALTER TABLE " + tName + " ADD " + refTableVariable + " INT NOT NULL";
            } else {
                query = "ALTER TABLE " + tName + " ADD COLUMN " + refTableVariable + " INT NOT NULL";
            }
            st.executeUpdate(query);
            query = "ALTER TABLE " + tName + " ADD CONSTRAINT " + "fkom" + tName + tRefName + " FOREIGN KEY (" + refTableVariable + ")" + " REFERENCES
            st.executeUpdate(query);
        } catch (SQLException e) {
            throw new IllegalStateException("DataBaseAutoGenerator1.createRelation1xN: Error", e);
        }
    }
    //……//

    //--Create relation nxm cbetween tables in the data base------------------//
    static void createRelationNxM(Connection connection, String tName1, String tName2) {
        try {
            Statement st = connection.createStatement();
            //--Create reference table-----------------------------------------//
            String variablefk1 = "fktref" + tName1 + tName2;
            String variablefk2 = "fktref" + tName2 + tName1;
            String variableId = "idtref" + tName1 + tName2;
            String constraint1 = "fknmtref" + tName1 + tName2; // + tName2
            String constraint2 = "fknmtref" + tName2 + tName1; // + tName1
            String query = "CREATE TABLE TREF"
                    + tName1 + tName2
                    + " ("
                    + variableId + " INT PRIMARY KEY, "
                    + variablefk1 + " INT NOT NULL, "
```

```
                + variablefk2 + " INT NOT NULL, "
                + "CONSTRAINT " + constraint1 + " FOREIGN KEY (" + variablefk1 + ") REFERENCES " + tName1 + "(id" + tName1 + ") ON DELETE CASCADE,
                + "CONSTRAINT " + constraint2 + " FOREIGN KEY (" + variablefk2 + ") REFERENCES " + tName2 + "(id" + tName2 + ") ON DELETE CASCADE)
        //------------------------------------------------------------------//
        st.executeUpdate(query);
    } catch (SQLException e) {
        throw new IllegalStateException("DataBaseAutoGenerator1.createRelationNxM: Error", e);
    }
}
//------------------------------------------------------------------//

//__Change Auto Increments from tables_____________________________________//
//--Data Banks have problems with auto inc, so I decided to remove --------//
static public void changeAutoIncFromTable(Connection connection, int i, int tableNumber) {
    try {
        Statement statement = connection.createStatement();
        String query = "ALTER TABLE";
        switch (i) {
            //__MySQL Option_____________________________________//
            case 0:
                String queryCreateIDMySQL = "ALTER TABLE" + " t" + tableNumber + " ADD COLUMN id" + "t" + tableNumber + " INT";
                statement.executeUpdate(queryCreateIDMySQL);
                query = query + " t" + tableNumber + " CHANGE id" + "t" + tableNumber + " id" + "t" + tableNumber + " INT PRIMARY KEY"; // INT AUT
                statement.executeUpdate(query);
                break;
            //__PostgreSQL Option_____________________________________//
            case 1:
                String queryCreateIDPostgreSQL = "ALTER TABLE" + " t" + tableNumber + " ADD COLUMN id" + "t" + tableNumber + " INT";
                statement.executeUpdate(queryCreateIDPostgreSQL);
                query = query + " t" + tableNumber + " ADD PRIMARY KEY (" + "idt" + tableNumber + ")"; // query = query + " t" + tableNumber + " A
                statement.executeUpdate(query);
                break;
            //__Firebird Option_____________________________________//
            case 2:
                String queryCreateFireBird = "ALTER TABLE" + " t" + tableNumber + " ADD id" + "t" + tableNumber + " INT PRIMARY KEY";
                statement.executeUpdate(queryCreateFireBird);
            /*
                String querySeq = "t" + tableNumber + "_seq";
                String queryGen = "CREATE GENERATOR " + querySeq;
                statement.executeUpdate(queryGen);
                String queryTrigger = "CREATE TRIGGER " + "trig_" + "t" + tableNumber
                        + " FOR " + "t" + tableNumber
                        + " BEFORE INSERT position 0 AS BEGIN new.id" + "t" + tableNumber + " = gen_id(" + querySeq + ",1); END;";
                statement.executeUpdate(queryTrigger);
             */
        }

    } catch (SQLException e) {
        throw new IllegalStateException("DataBaseAutoGenerator1.changeAutoIncFromTable: Error", e);
    }

}

//**Drop  Functions*************************************************//
//__Just to continue Execute funtion_____________________________________//
static public double dropLoop(int database, List<String> tables, List<String> listForeignKeys) {
    double elapsedTimeDrop = 0;
    try (Connection connection = DriverManager.getConnection(dbConnection.getUrl().get(database), dbConnection.getUser().get(database), dbConnecti
        switch (database) {
            case 0:
                tables = dbConnection.getTablesMySQL(database);
                break;
            case 1:
                tables = dbConnection.getTablesPostgreSQL(database);
                break;
            case 2:
                tables = dbConnection.getTablesFireBird(database);
        }

        for (int i = 0; i < tables.size(); i++) {
            listForeignKeys = getForeignKey(connection, tables.get(i), 3);
            for (int j = 0; j < listForeignKeys.size(); j++) {
                removeForeignKey(connection, tables.get(i), listForeignKeys.get(j));
            }
        }
        long  startDrop  =  System.nanoTime();
        dropTableDataBase(connection, tables);
        long finishDrop = System.nanoTime() - startDrop;
        elapsedTimeDrop = (double) finishDrop / 1000000;
        connection.close();
    } catch (SQLException e) {
        throw new IllegalStateException("DataBaseAutoGenerator1.Main:error", e);
    }
    return elapsedTimeDrop;
}

//------------------------------------------------------------------//
//__Drop all the tables in a database_____________________________________//
static public void dropTableDataBase(Connection connection, List<String> tables) throws SQLException {
    connection.setAutoCommit(false);
    Statement statement = connection.createStatement();
    for (int i = 0; i < tables.size(); i++) {
        String query = "DROP TABLE";
```

```
            query = query + " " + tables.get(i);
            statement.addBatch(query);
        }
        statement.executeBatch();
        connection.commit();
    }

    //.....//
    public static void dropTriggerDataBase(Connection connection, String tName) {
        String query = "";
        try {
            Statement statement = connection.createStatement();
            query = "DROP TRIGGER " + "TRIG_" + tName;
            statement.executeUpdate(query);
        } catch (SQLException e) {
            return;
        }
    }
    //.....//

    //.....//
    public static void dropGeneratorDataBase(Connection connection, String tName) {
        String query = "";
        try {
            Statement statement = connection.createStatement();
            query = "DROP GENERATOR " + tName + "_SEQ";
            statement.execute(query);
        } catch (SQLException e) {
            return;
        }
    }
    //.....//
    //*********************************************************************//

    //--Remove foreign key from a table----------------------------------------//
    public static void removeForeignKey(Connection connection, String tName, String relation) {
        try {
            Statement st = connection.createStatement();
            String query = "";
            String conName = connection.toString();

            //--Verify if is MySQL because of sintax--------------------------//
            if (conName.contains("mysql")) {
                query = "ALTER TABLE " + tName + " DROP FOREIGN KEY ";
            } else {
                query = "ALTER TABLE " + tName + " DROP CONSTRAINT ";

            }
            //----------------------------------------------------------------//

            if (tName.contains("tref") || tName.contains("TREF")) {
                String queryAux = query;
                try {
                    queryAux = queryAux + "fknmtref" + tName.substring(4, tName.length());
                    st.executeUpdate(queryAux);
                } catch (SQLException e) {
                    queryAux = query;
                    String varNameAux = "fknmtref";
                    for (int i = 5; i < tName.length(); i++) {
                        if ((tName.charAt(i) == 't') || (tName.charAt(i) == 'T')) {
                            varNameAux = varNameAux + tName.substring(i, tName.length());
                            break;
                        }
                    }
                    varNameAux = varNameAux + "t";
                    for (int i = 5; i < tName.length(); i++) {
                        if ((tName.charAt(i) == 't') || (tName.charAt(i) == 'T')) {
                            break;
                        }
                        varNameAux = varNameAux + tName.charAt(i);
                    }
                    queryAux = queryAux + varNameAux;
                    st.executeUpdate(queryAux);
                }
            } else {
                String queryAux = query;
                try {
                    queryAux = queryAux + "fkoo" + tName + relation;
                    st.executeUpdate(queryAux);
                } catch (SQLException e) {
                    queryAux = query;
                    queryAux = queryAux + "fkom" + tName + relation;
                    st.executeUpdate(queryAux);
                }
            }
        } catch (SQLException e) {
            throw new IllegalStateException("DataBaseAutoGenerator1.createRelationNxM: Error", e);
        }
    }

    //.....//
    //.....//
    public static List<String> getForeignKey(Connection connection, String tableName, int i) throws SQLException {
```

```
        List<String> fkTableData = new ArrayList<>();
        DatabaseMetaData dm =  connection.getMetaData();
        ResultSet rs = dm.getImportedKeys(null, null, tableName);
        while (rs.next()) {
            fkTableData.add(rs.getString(i));
        }
        return fkTableData;
    }
    //-------------------------------------------------------------------------//

    //__Create E/R model_____________________________________________________//
    public static void createERModel(int database, List<Pair<String, Pair<String, String>>> listRelations) {
        try (Connection connection = DriverManager.getConnection(dbConnection.getUrl().get(database), dbConnection.getUser().get(database), dbConnecti
            System.out.println("Stablishing relations between tables");
            String option = "";
            String parameter1 = "";
            String parameter2 = "";

            for (int i = 0; i < listRelations.size(); i++) {
                option = listRelations.get(i).getKey();
                parameter1 = listRelations.get(i).getValue().getKey();
                parameter2 = listRelations.get(i).getValue().getValue();
                switch (option) {
                    case "oo":
                        createRelation1x1(connection, parameter1, parameter2, "id" + parameter1, "id" + parameter2);
                        break;
                    case "om":
                        createRelation1xN(connection, parameter1, parameter2, "id" + parameter2);
                        break;
                    case "nm":
                        createRelationNxM(connection, parameter1, parameter2);
                }
            }
            connection.close();
        } catch (SQLException e) {
            throw new IllegalStateException("DataBaseAutoGenerator1.Main:error", e);
        }
    }
    //-------------------------------------------------------------------------//
}
```

C – Manipulador de Base de Dados

```
package apptcc;

import javafx.util.Pair;
import java.sql.Connection;
import java.sql.DatabaseMetaData;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;
import java.time.Duration;
import java.time.Instant;
import java.util.ArrayList;
import java.util.List;
import java.util.ListIterator;
import java.util.Random;
import java.util.Scanner;
import java.util.Timer;
import java.util.concurrent.TimeUnit;

/**
 *
 * @author Rafhael
 */
public class DataBaseAutoM {

    //__General use_____________________________________________________________//
    private static Scanner scanner = new Scanner(System.in);
    private static Random rand = new Random(213457624L);
    private static DBConnection dbConnection = new DBConnection();
    private static List<List<Object>> functionMessages;
    private static List<Double> dataList;
    //__________________________________________________________________________//

    public static Pair<List<Double>, List<List<Object>>> executeManipulator(int ins, int sel, int up, int del, int database, List<Integer> opt, boolea

        //--List to return with data collected-----------------------------------//
        dataList = new ArrayList<>();
        functionMessages = new ArrayList();
        //__________________________________________________________________________//

        //--variables for database manipulation----------------------------------//
        List<String> tName = new ArrayList<>();
        List<List<String>> variablesName = new ArrayList<>();
        List<List<String>> variablesType = new ArrayList<>();
        List<Pair<String, Pair<List<String>, List<String>>>> listVar;
        int dataQuantInsert = ins; // quantity of data to insert
        int dataQuantUpdate = up; // quantity of data to update
        int dataQuantDelete = del;
        //__________________________________________________________________________//

        //__Apply operations_______________________________________________________//
        List<Integer> option = opt;

        //for (int databases = 0; databases <= 2; databases++) {
        try (Connection connection = DriverManager.getConnection(dbConnection.getUrl().get(database), dbConnection.getUser().get(database), dbConnecti

            //__Get Tables names_____________________________________________//
            switch (database) {
                case 0:
                    tName = dbConnection.getTablesMySQL(database);
                    break;
                case 1:
                    tName = dbConnection.getTablesPostgreSQL(database);
                    break;
                case 2:
                    tName = dbConnection.getTablesFireBird(database);
            }
            //__________________________________________________________________//

            //__Get Tables info______________________________________________//
            for (int j = 0; j < tName.size(); j++) {
                variablesName.add(getVariablesName(connection, tName.get(j)));
                variablesType.add(getVariablesType(connection, tName.get(j)));
            }
            //__________________________________________________________________//

            //--Get a new list for database access-----------------------//
            listVar = new ArrayList<>();
            for (int i = 0; i < tName.size(); i++) {
                Pair<String, Pair<List<String>, List<String>>> pairAux = new Pair(tName.get(i), new Pair(variablesName.get(i), variablesType.get(i)));
                listVar.add(pairAux);
            }
            //__________________________________________________________________//
            connection.close();
        } catch (SQLException e) {
```

```
            throw new IllegalStateException("DataBaseManipulator.Main:error", e);
        }

        for (int op = 0; op < option.size(); op++) {
            switch (option.get(op)) {
                case 1:
                    dataList.add(insertLoop(dataQuantInsert, listVar, SHOW_PRINT_MESSAGES, database, BATCH_PROCESSING));
                    break;

                case 2:
                    dataList.add(selectLoop(listVar, SHOW_PRINT_MESSAGES, database));
                    break;

                case 4:
                    dataList.add(updateLoop(dataQuantUpdate, listVar, SHOW_PRINT_MESSAGES, database, BATCH_PROCESSING));
                    break;

                case 5:
                    dataList.add(deleteLoop1(dataQuantDelete, listVar, SHOW_PRINT_MESSAGES, database, BATCH_PROCESSING));
            }
        }
        return new Pair(dataList, functionMessages);
    }
    //*********************************************************************//

    //**Functions Just to Continue Execution Loop*************************//
    public static double insertLoop(int dataQuantInsert, List<Pair<String, Pair<List<String>, List<String>>>> listVar, boolean SHOW_PRINT_MESSAGES, in
        List<Double> addTimeInsert = new ArrayList<>();
        Connection connectionInsert = DriverManager.getConnection(dbConnection.getUrl().get(database), dbConnection.getUser().get(database), dbConnect
        for (int table = 0; table < listVar.size(); table++) {
            long startInsert = System.nanoTime();
            insert(connectionInsert, listVar.get(table).getKey(), listVar.get(table).getValue().getKey(), listVar.get(table).getValue().getValue(), da
            long finishInsert = System.nanoTime() - startInsert;
            double elapsedTimeInsert = (double) finishInsert / 1000000;;
            addTimeInsert.add(elapsedTimeInsert);
        }
        double elapsedTimeInsertMS = 0;
        for (int i = 0; i < addTimeInsert.size(); i++) {
            elapsedTimeInsertMS += addTimeInsert.get(i);
        }
        connectionInsert.close();
        return elapsedTimeInsertMS;
    }

    public static double selectLoop(List<Pair<String, Pair<List<String>, List<String>>>> listVar, boolean SHOW_PRINT_MESSAGES, int database) {
        double elapsedTimeSelectAllMS = 0;


        try (Connection connection = DriverManager.getConnection(dbConnection.getUrl().get(database), dbConnection.getUser().get(database), dbConnecti
            for (int j = 0; j < listVar.size(); j++) {
                List<Object> selectMessages = new ArrayList<>();
                long startSelectAll = System.nanoTime();
                List<List<Object>> rs = select(connection, listVar.get(j));
                long finishSelectAll = System.nanoTime() - startSelectAll;
                elapsedTimeSelectAllMS = (double) finishSelectAll / 1000000;
                for (int rsi = 0; rsi < rs.size(); rsi++) {
                    Object obj = "";
                    for (int rsj = 0; rsj < rs.get(rsi).size(); rsj++) {
                        obj += "[" + (String) rs.get(rsi).get(rsj).toString() + "] ";
                    }
                    selectMessages.add(obj);
                }
                functionMessages.add(selectMessages);
            }
            connection.close();
        } catch (SQLException e) {
            throw new IllegalStateException("DataBaseManipulator.Main:error", e);
        }
        return elapsedTimeSelectAllMS;
    }

    public static double updateLoop(int dataQuantUpdate, List<Pair<String, Pair<List<String>, List<String>>>> listVar, boolean SHOW_PRINT_MESSAGES, in
        Connection connectionUpdate = DriverManager.getConnection(dbConnection.getUrl().get(database), dbConnection.getUser().get(database), dbConnect
        List<Double> listelapsedTimeUpdateMS = new ArrayList<>();
        long elapsedTimeUpdateMSAux = 0;
        for (int table = 0; table < listVar.size(); table++) {
            int element = 5;
            int index = switchF(listVar.get(table).getValue().getValue().get(element));
            long startUpdate = System.nanoTime();
            switch (index) {
                case 1:
                    update(connectionUpdate, listVar.get(table).getKey(), listVar.get(table).getValue().getKey().get(element), listVar.get(table).getV
                    break;
                case 2:
                    update(connectionUpdate, listVar.get(table).getKey(), listVar.get(table).getValue().getKey().get(element), listVar.get(table).getV
                    break;
                case 3:
                    update(connectionUpdate, listVar.get(table).getKey(), listVar.get(table).getValue().getKey().get(element), listVar.get(table).getV
                    break;
                case 4:
                    update(connectionUpdate, listVar.get(table).getKey(), listVar.get(table).getValue().getKey().get(element), listVar.get(table).getV
            }
            long finishUpdate = System.nanoTime() - startUpdate;
```

```
            double elapsedTimeUpdate = (double) finishUpdate / 1000000;
            listelapsedTimeUpdateMS.add(elapsedTimeUpdate);
        }
        double elapsedTimeUpdateMS = 0;
        for (int i = 0; i < listelapsedTimeUpdateMS.size(); i++) {
            elapsedTimeUpdateMS += listelapsedTimeUpdateMS.get(i);
        }
        connectionUpdate.close();
        return elapsedTimeUpdateMS;
    }

    public static double deleteLoop1(int dataQuantDelete, List<Pair<String, Pair<List<String>, List<String>>>> listVar, boolean SHOW_PRINT_MESSAGES, i
        Connection connectionDelete = DriverManager.getConnection(dbConnection.getUrl().get(database), dbConnection.getUser().get(database), dbConnect
        List<Double> listDeleteTime = new ArrayList();
        for (int table = 0; table < listVar.size(); table++) {
            long startDelete = System.nanoTime();
            int result = delete(connectionDelete, listVar.get(table).getKey(), dataQuantDelete, SHOW_PRINT_MESSAGES, BATCH_PROCESSING);
            long finishDelete = System.nanoTime() - startDelete;
            double elapsedTimeDelete = (double) finishDelete / 1000000; //--get miliseconds
            listDeleteTime.add(elapsedTimeDelete);
        }
        double elapsedTimeDeleteMS = 0;
        for (int i = 0; i < listDeleteTime.size(); i++) {
            elapsedTimeDeleteMS += listDeleteTime.get(i);
        }
        connectionDelete.close();
        return elapsedTimeDeleteMS;
    }
//**********************************************************************//

    //**Get Elements at tables************************************************//
    //.....................................................................//
    public static List<String> getVariablesName(Connection connection, String tName) {
        List<String> variable = new ArrayList<>();
        try {
            Statement st = connection.createStatement();
            ResultSet rs = st.executeQuery("SELECT * FROM " + tName);
            ResultSetMetaData rsmd = rs.getMetaData();
            int columnCount = rsmd.getColumnCount();
            String varName = "";
            for (int i = 1; i <= columnCount; i++) {
                varName = rsmd.getColumnName(i);
                variable.add(varName);
            }
        } catch (SQLException e) {
            throw new RuntimeException(e);

        }
        return variable;
    }
    //.....................................................................//

    //.....................................................................//
    public static List<String> getVariablesType(Connection connection, String tName) {
        List<String> variable = new ArrayList<>();
        try {
            Statement st = connection.createStatement();
            ResultSet rs = st.executeQuery("SELECT * FROM " + tName);
            ResultSetMetaData rsmd = rs.getMetaData();
            int columnCount = rsmd.getColumnCount();
            String varType = "";
            for (int i = 1; i <= columnCount; i++) {
                varType = rsmd.getColumnTypeName(i);
                variable.add(varType);
            }
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
        return variable;
    }
    //.....................................................................//
    //**********************************************************************//

    //--Select all from a table-------------------------------------------//
    public static List<List<Object>> select(Connection connection, Pair<String, Pair<List<String>, List<String>>> table) {
        String query = "";
        List<Object> selectResultAux;
        List<List<Object>> selectResult = new ArrayList<>();
        try {
            Statement statement = connection.createStatement();
            query = "SELECT * FROM " + table.getKey();
            ResultSet result = statement.executeQuery(query);
            while (result.next()) {
                selectResultAux = new ArrayList<>();
                for (int i = 0; i < table.getValue().getValue().size(); i++) {
                    int rs = switchF(table.getValue().getValue().get(i));
                    switch (rs) {
                        case 1:
                            selectResultAux.add(result.getInt(table.getValue().getKey().get(i)));
                            break;
                        case 2:
                            selectResultAux.add(result.getString(table.getValue().getKey().get(i)));
                            break;
```

```
                    case 3:
                        selectResultAux.add(result.getFloat(table.getValue().getKey().get(i)));
                        break;
                    case 4:
                        selectResultAux.add(result.getDate(table.getValue().getKey().get(i)));
                }
            }
            selectResult.add(selectResultAux);
        }
    } catch (SQLException e) {
        throw new IllegalStateException("DataBaseManipulator.selectAll: function error ", e);
    }
    return selectResult;
}
//-------------------------------------------------------------------------//

//-- Insert Method-----------------------------------------------------------//
public static void insert(Connection connection, String tName, List<String> variablesName, List<String> variablesType, int dataQuantInsert, boolea
    //-- Option only for batch processing-----------------------------------//
    if (BATCH_PROCESSING == true) {
        connection.setAutoCommit(false);
    }
    //-------------------------------------------------------------------------//
    List<Object> insertMessageList = new ArrayList<>();
    String query = "";
    List<String> variablesNameAux = new ArrayList<String>(variablesName);
    List<String> variablesTypeAux = new ArrayList<String>(variablesType);
    try {
        query = "INSERT INTO " + tName + " (";
        //-- build insert string-----------------------------------------------//
        for (int i = 0; i < variablesNameAux.size(); i++) {
            query = query + variablesNameAux.get(i) + ",";
        }

        StringBuilder sbuilder = new StringBuilder(query);
        sbuilder.setCharAt(query.length() - 1, ')');
        query = sbuilder.toString();

        query = query + " VALUES (";

        for (int u = 0; u < variablesTypeAux.size(); u++) {
            query = query + "?,";
        }

        sbuilder = new StringBuilder(query);
        sbuilder.setCharAt(query.length() - 1, ')');

        query = sbuilder.toString();
        //-------------------------------------------------------------------------//

        //-- insert values in the string---------------------------------------//
        PreparedStatement pStatement = connection.prepareStatement(query);
        for (int keysValues = 1; keysValues <= dataQuantInsert; keysValues++) {
            int counter = 1;
            for (int k = 0; k < variablesTypeAux.size(); k++) {
                if (variablesNameAux.get(k).contains("id") || variablesNameAux.get(k).contains("ID") || variablesNameAux.get(k).contains("fk") ||
                    pStatement.setInt(counter, keysValues);
                } else {
                    int index = switchF(variablesTypeAux.get(k));
                    switch (index) {
                        case 1:
                            pStatement.setInt(counter, 1111111111);
                            break;
                        case 2:
                            pStatement.setString(counter, "BENCHMARK TESTING");
                            break;
                        case 3:
                            pStatement.setFloat(counter, 222222222);
                            break;
                        case 4:
                            pStatement.setDate(counter, java.sql.Date.valueOf("2020-02-11"));
                    }
                }
                counter++;
            }
            //--Execute single query or batch------------------------------//
            if (BATCH_PROCESSING == false) {
                pStatement.executeUpdate();
            }
            if (BATCH_PROCESSING == true) {
                pStatement.addBatch();
            }
            //-------------------------------------------------------------------//

            insertMessageList.add("Values id (" + keysValues + ") inserted in (" + tName + ")");

        }
        if (BATCH_PROCESSING == true) {
            pStatement.executeBatch();
            connection.commit();
        }
        functionMessages.add(insertMessageList);
    } catch (SQLException e) {
```

```
            System.out.println(e);
        }
    }
    //*************************************************************************//

    //--Delete Batch of Elements from table-----------------------------------//
    public static int delete(Connection connection, String tName, int quantDelete, boolean SHOW_PRINT_MESSAGES, boolean BATCH_PROCESSING) throws SQLEx
        //--Option to execute batch processing-----------------------------//
        if (BATCH_PROCESSING == true) {
            connection.setAutoCommit(false);
        }
        //-----------------------------------------------------------------//
        String query = "";
        try {
            query = "DELETE FROM " + tName + " WHERE id" + tName + " = ?";
            PreparedStatement ps = connection.prepareStatement(query);
            List<Object> deleteMessageList = new ArrayList<>();
            for (int id = 1; id <= quantDelete; id++) {
                ps.setInt(1, id);
                //--Execute single query or batch----------------------------//
                if (BATCH_PROCESSING == true) {
                    ps.addBatch();
                } else {
                    ps.executeUpdate();
                }
                //-----------------------------------------------------------//
                String deleteMessage = "Line (" + id + ") Table (" + tName + ")" + " Deleted";
                deleteMessageList.add(deleteMessage);
            }
            ps.executeBatch();
            connection.commit();
            functionMessages.add(deleteMessageList);
            return 0;
        } catch (SQLException e) {
            return 1;
        }
    }
    //-----------------------------------------------------------------------//

    //--Update Batch of Elements from table-----------------------------------//
    public static int update(Connection connection, String tName, String variableName, String variableType, String variableValue, int quantUpdate, boo
        if (BATCH_PROCESSING == true) {
            connection.setAutoCommit(false);
        }
        String query = "";
        List<Object> updateMessageList = new ArrayList<>();

        try {
            query = "UPDATE " + tName + " SET " + variableName + " = ?" + " WHERE id" + tName + " = ?";
            PreparedStatement ps = connection.prepareStatement(query);
            for (int id = 1; id <= quantUpdate; id++) {
                int index = switchF(variableType);
                switch (index) {
                    case 1:
                        ps.setInt(1, Integer.parseInt(variableValue));
                        break;
                    case 2:
                        ps.setString(1, "" + variableValue);
                        break;
                    case 3:
                        ps.setFloat(1, Float.parseFloat(variableValue));
                        break;
                    case 4:
                        ps.setDate(1, java.sql.Date.valueOf(variableValue));
                        break;
                }
                ps.setInt(2, id);
                //--Execute single query or batch----------------------------//
                if (BATCH_PROCESSING == true) {
                    ps.addBatch();
                } else {
                    ps.executeUpdate();
                }
                //-----------------------------------------------------------//

                updateMessageList.add("Line (" + id + ") Table (" + tName + ")"
                        + " Variable (" + variableName + ")"
                        + " Type (" + variableType + ")"
                        + " Updated in id = " + id);

            }
            if (BATCH_PROCESSING == true) {
                ps.executeBatch();
                connection.commit();
            }
            functionMessages.add(updateMessageList);
            return 0;
        } catch (SQLException e) {
            return 1;
        }
    }
    //-----------------------------------------------------------------------//
```

```
    //--Gather all sintaxes of data banks variables---------------------------//
    private static int switchF(String variableType) {
        switch (variableType) {
            case "INT":
                return 1;
            case "INTEGER":
                return 1;
            case "int4":
                return 1;
            case "VARCHAR":
                return 2;
            case "varchar":
                return 2;
            case "FLOAT":
                return 3;
            case "float8":
                return 3;
            case "DATE":
                return 4;
            case "date":
                return 4;
        }
        return 0;
    }
    //-----------------------------------------------------------------------//
}
```

D – Algoritmo de Escrita no Excel

```
package apptcc;

/**
 *
 * @author Rafhael
 */

import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

public class WriterExcel {

    public void write(Object[][] datatypes, String FILE_NAME) {

        XSSFWorkbook workbook = new XSSFWorkbook();
        XSSFSheet sheet = workbook.createSheet("Datatypes in Java");

        int rowNum = 0;
        for (Object[] datatype : datatypes) {
            Row row = sheet.createRow(rowNum++);
            int colNum = 0;
            for (Object field : datatype) {
                Cell cell = row.createCell(colNum++);
                if (field instanceof String) {
                    cell.setCellValue((String) field);
                } else if (field instanceof Integer) {
                    cell.setCellValue((Integer) field);
                } else if (field instanceof Long){
                    cell.setCellValue((Long) field);
                }else if (field instanceof Double){
                    cell.setCellValue((Double) field);
                }
            }
        }
```

```
        try {
            FileOutputStream outputStream = new FileOutputStream(FILE_NAME)
            workbook.write(outputStream);
            workbook.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

}
```

E - Algoritmo de Escrita em Arquivo TXT

```
/*
 * To change this license header, choose License Headers in Project Propert
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package apptcc;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;

/**
 *
 * @author Rafhael
 */
public class WriterTXT {
    public void write(String line) {
        try {
            FileWriter wr = new FileWriter("C:/Users/Rafhael/Desktop/TCC/Re
            BufferedWriter bw = new BufferedWriter(wr);
            bw.write(line);
            bw.newLine();
            bw.close();
        } catch (IOException e) {
            throw new IllegalStateException("TXTManipulator.write:error");
        }
    }
}
```

F – Classe de Conexão com os Bancos de Dados

```
package apptcc;

import java.sql.Connection;
import java.sql.DatabaseMetaData;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import java.util.ListIterator;
import sun.security.util.Password;

/**
 *
 * @author Rafhael
 */
public class DBConnection {

    private static List<String> url;
    private static List<String> user;
    private static List<String> password;
    private static Connection connection;

    public DBConnection() {
        url = new ArrayList<>();
        url.add("jdbc:mysql://localhost:3308/teste1?serverTimezone=UTC");
        url.add("jdbc:postgresql://localhost:5432/teste1");
        url.add("jdbc:firebirdsql:localhost/3050:C:/Users/Rafhael/Desktop/T
        user = new ArrayList<>();
        user.add("root");
        user.add("postgres");
        user.add("SYSDBA");
        password = new ArrayList<>();
        password.add("");
        password.add("");
        password.add("");
    }

    public List<String> getUrl() {
```

```
        return url;
    }

    public void setUrl(List<String> url) {
        this.url = url;
    }

    public List<String> getUser() {
        return user;
    }

    public void setUser(List<String> user) {
        this.user = user;
    }

    public List<String> getPassword() {
        return password;
    }

    public void setPassword(List<String> password) {
        this.password = password;
    }

     //---Works only for MySQL-----------------------------------------------
    public static List<String> getTablesMySQL(int database) {
        List<String> tName = new ArrayList<>();
        try {
            connection = DriverManager.getConnection(url.get(database), use
            DatabaseMetaData metaData = connection.getMetaData();
            ResultSet rs = metaData.getTables("teste1", null, "%", null);
            while (rs.next()) {
                tName.add(rs.getString(3));
            }
            connection.close();
        } catch (SQLException e) {
            throw new IllegalStateException("DataBaseAutoGenerator1.getTabl
        }
        return tName;
    }
    //--------------------------------------------------------------------------

    //---Works only for PostgreSQL-----------------------------------------
    public static List<String> getTablesPostgreSQL(int database) {

        List<String> tName = new ArrayList<>();
        List<String> tNameAux = new ArrayList<>();
```

```
    try {
            connection = DriverManager.getConnection(url.get(database), use
            DatabaseMetaData metaData = connection.getMetaData();
            ResultSet rs = metaData.getTables("teste1", null, "%", null);
            while (rs.next()) {
                tName.add(rs.getString(3));
            }
            connection.close();
        } catch (SQLException e) {
            throw new IllegalStateException("DataBaseAutoGenerator1.getTabl
        }
        for (ListIterator<String> iter = tName.listIterator(); iter.hasNext
            String element = iter.next();
            if ((element.charAt(0) == 't' && ((element.charAt(element.lengt
                    || (element.charAt(element.length() - 1) == '1')
                    || (element.charAt(element.length() - 1) == '2')
                    || (element.charAt(element.length() - 1) == '3')
                    || (element.charAt(element.length() - 1) == '4')
                    || (element.charAt(element.length() - 1) == '5')
                    || (element.charAt(element.length() - 1) == '6')
                    || (element.charAt(element.length() - 1) == '7')
                    || (element.charAt(element.length() - 1) == '8')
                    || (element.charAt(element.length() - 1) == '9')))) {
                tNameAux.add(element);
            }
        }
        return tNameAux;
    }
    //-------------------------------------------------------------------

    //---Works only for FireBird-----------------------------------------
    public static List<String> getTablesFireBird(int database) {
        List<String> tName = new ArrayList<>();
        List<String> tNameAux = new ArrayList<>();
        try {
            connection = DriverManager.getConnection(url.get(database), use
            DatabaseMetaData metaData = connection.getMetaData();
            ResultSet rs = metaData.getTables("teste1", null, "%", null);
            while (rs.next()) {
                tName.add(rs.getString(3));
            }
            connection.close();
        } catch (SQLException e) {
            throw new IllegalStateException("DataBaseAutoGenerator1.getTabl
        }
        for (ListIterator<String> iter = tName.listIterator(); iter.hasNext
```

```
    try {
            connection = DriverManager.getConnection(url.get(database), use
            DatabaseMetaData metaData = connection.getMetaData();
            ResultSet rs = metaData.getTables("teste1", null, "%", null);
            while (rs.next()) {
                tName.add(rs.getString(3));
            }
            connection.close();
        } catch (SQLException e) {
            throw new IllegalStateException("DataBaseAutoGenerator1.getTabl
        }
        for (ListIterator<String> iter = tName.listIterator(); iter.hasNext
            String element = iter.next();
            if ((element.charAt(0) == 't' && ((element.charAt(element.lengt
                    || (element.charAt(element.length() - 1) == '1')
                    || (element.charAt(element.length() - 1) == '2')
                    || (element.charAt(element.length() - 1) == '3')
                    || (element.charAt(element.length() - 1) == '4')
                    || (element.charAt(element.length() - 1) == '5')
                    || (element.charAt(element.length() - 1) == '6')
                    || (element.charAt(element.length() - 1) == '7')
                    || (element.charAt(element.length() - 1) == '8')
                    || (element.charAt(element.length() - 1) == '9')))) {
                tNameAux.add(element);
            }
        }
        return tNameAux;
    }
    //-------------------------------------------------------------------

    //---Works only for FireBird-----------------------------------------
    public static List<String> getTablesFireBird(int database) {
        List<String> tName = new ArrayList<>();
        List<String> tNameAux = new ArrayList<>();
        try {
            connection = DriverManager.getConnection(url.get(database), use
            DatabaseMetaData metaData = connection.getMetaData();
            ResultSet rs = metaData.getTables("teste1", null, "%", null);
            while (rs.next()) {
                tName.add(rs.getString(3));
            }
            connection.close();
        } catch (SQLException e) {
            throw new IllegalStateException("DataBaseAutoGenerator1.getTabl
        }
        for (ListIterator<String> iter = tName.listIterator(); iter.hasNext
```

```
            String element = iter.next();
            if ((element.charAt(0) == 'T' && ((element.charAt(element.lengt
                    || (element.charAt(element.length() - 1) == '1')
                    || (element.charAt(element.length() - 1) == '2')
                    || (element.charAt(element.length() - 1) == '3')
                    || (element.charAt(element.length() - 1) == '4')
                    || (element.charAt(element.length() - 1) == '5')
                    || (element.charAt(element.length() - 1) == '6')
                    || (element.charAt(element.length() - 1) == '7')
                    || (element.charAt(element.length() - 1) == '8')
                    || (element.charAt(element.length() - 1) == '9')))) {
                tNameAux.add(element);
            }
        }
        return tNameAux;
    }
    //---------------------------------------------------------------------
}
```

G – Exemplo de uma Base de Dados com o Modelo E/R

A figura a seguir mostra uma base de dados gerada pelo algoritmo de criação de base de dados para testes envolvendo o modelo E/R:

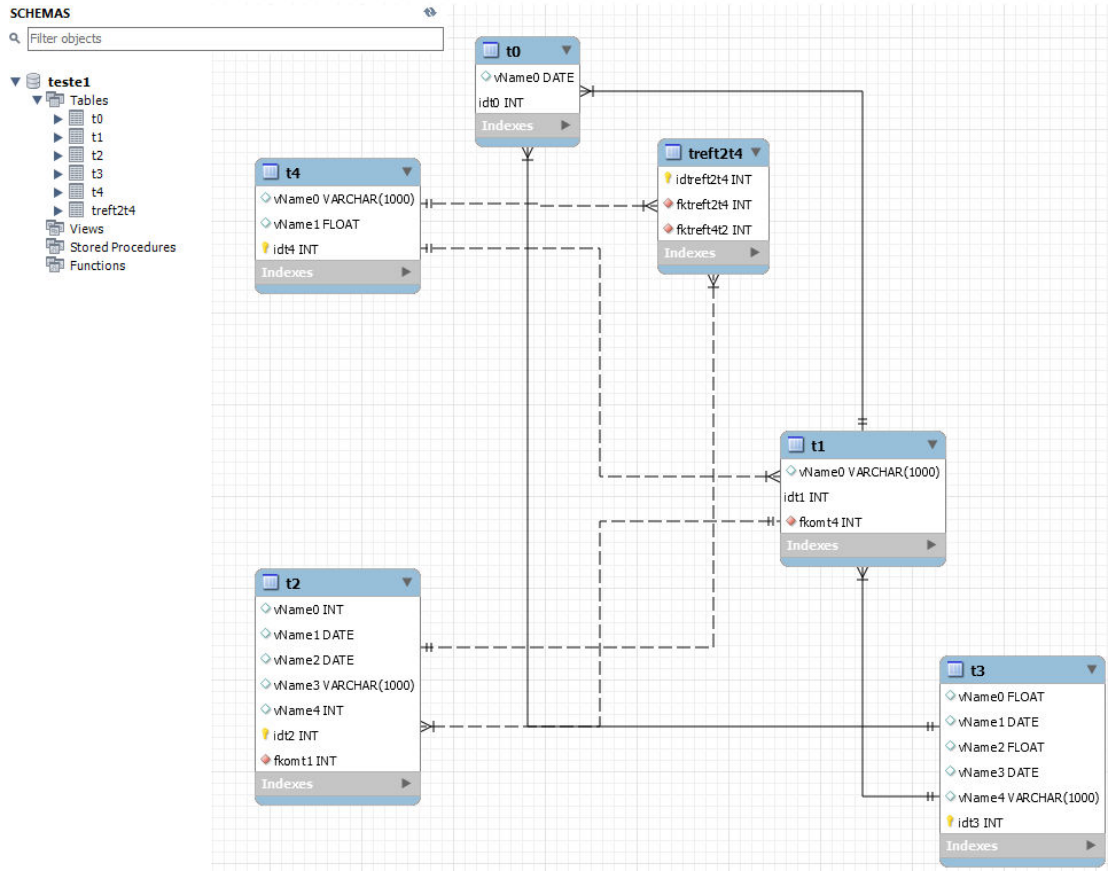


Figura 37 - Exemplo de Base de Dados com o Modelo E/R

Fonte – Autoria Própria

A figura 37 foi gerada a partir do próprio *Workbench* do MySQL depois do algoritmo de criação de base de dados gerar o modelo no banco, o *software* é capaz de gerar uma base de dados de qualquer tamanho e simular todas as relações em um modelo E/R.

H – Gráficos Realizados com Apenas uma Iteração

A figura a seguir mostra o resultado dos testes realizados com o *software* realizando uma única iteração, foram utilizadas as mesmas quantidades de operações dos testes com pequenos volumes de dados porque refletem melhor a dificuldade na obtenção dos resultados:

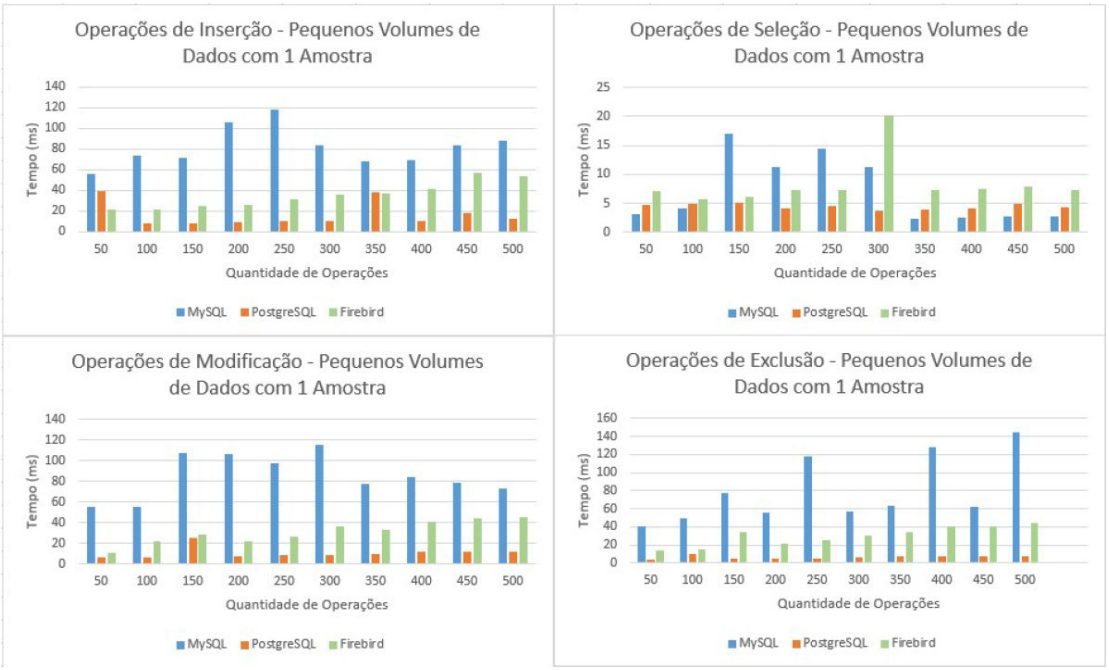


Figura 38 - Gráficos das Operações CRUD com uma Amostra

Fonte: Autoria Própria.

A partir da figura 38 é observa-se que não é possível obter resultados de desempenho por meio de uma única amostra, o algoritmo deve ser executado várias vezes para que seja estabelecida a média desses valores formando assim os gráficos de desempenho.