UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

DAVID MEES KNIJNIK

Comparação de SGBDs MongoDB e PostgreSQL para Jogos Digitais

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação

Orientadora: Profa. Dra. Renata Galante

Porto Alegre
2022

AGRADECIMENTOS

Gostaria de agradecer a minha família, em especial a minha mãe, Lúcia Alves Mees, pelo acompanhamento da redação deste texto e por ter estado sempre presente para mim, assim como meu pai, Denis Knijnik e o meu irmão, Leonardo Mees Knijnik. Também Gostaria de agradecer a minha orientadora, Renata Galante, que me ajudou durante todo o desenvolvimento do TCC, tendo sido fundamental para a aprendizagem e elaboração de meu escrito. Agradeço ainda aos meus amigos e colegas pelos bons momentos, o que me ajudou a relaxar nos intervalos do trabalho.

RESUMO

Bancos de dados são usados em grande parte das aplicações de jogos digitais. Existem várias opções para a escolha de Sistema de Gerenciamento de Banco de Dados para jogos. Jogos diferentes ainda têm necessidades diferentes para bancos de dados. Portanto, é importante para desenvolvedores estarem bem informados para escolherem um SGDB correto para um jogo sendo desenvolvido. Esse trabalho visa atender a essa necessidade realizando uma análise de SGDBs no contexto de jogos digitais. São citados jogos desenvolvidos anteriormente, junto com seus bancos de dado. Depois, é feita uma comparação entre o MongoDB e o PostgreSQL. O MongoDB teve um desempenho melhor na carga de dados novos e em filtragens simples de dados, enquanto que o PostgreSQL teve um melhor desempenho juntando dados de grupos diferentes e efetuando consultas mais complexas. Por isso, o trabalho constatou que o MongoDB é o melhor para coletar dados dos jogadores, enquanto que o PostgreSQL é melhor para gerir a lógica interna de servidores dedicados do jogo.

Palavras-chave: Jogos Digitais. Bancos de Dados. NoSQL. MongoDB. PostgreSQL.

Name: Analysis of DBMS used in Game Development

ABSTRACT

Databases are used in a lot of digital games. There are a lot of options for a Database Management System (DBMS) for games. Different games have different necessities for their databases. Because of this it's important for developers to be well informed to be able to choose the correct DBMS for the game they are developing. This work aims to meet that need by making an analysis of DBMSs in the context of game development. Previously developed games are cited, together with their databases. After that, comparison between MongoDB and PostgreSQL is made. MongoDB had a better performance loading and saving data and on simple selection queries, while PostgreSQL had better performance joining data from different groups and making complex queries. Because of that, this work concludes that MongoDB is better for player data collection, while PostgreSQL is better for managing the internal logic for dedicated servers.

Keywords: Digital Games. Databases. NoSQL. MongoDB. PostgreSQL.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CSV	*Comma-separated values,* tipo de arquivo no qual há valores separados por vírgulas.
API	*Application Programming Interface*
SQL	*Structured Query Language*
SGDB	*Sistema de Gerenciamento de Banco de Dados*
DBaaS	*Database as a Service*

SUMÁRIO

1 INTRODUÇÃO

Empresas de desenvolvimento de jogos eletrônicos têm usado cada vez mais bancos de dados. Qualquer jogo que faça uso de uma conexão com a internet, para que jogadores possam jogar juntos, precisa fazer uso de bancos de dados para guardar informações das contas dos jogadores. Se o jogo envolver partidas competitivas entre jogadores, ele pode também guardar, com segurança, dados da partida, de modo que não possam ser manipulados pelos jogadores diretamente.

Com o advento de tipos diferentes de sistemas de gerenciamento de bancos de dados (SGBDs), junto com ferramentas de Database As a Service (DBaaS), que são serviços de aluguel de recursos de armazenamento de dados por um preço, o começo do desenvolvimento de qualquer jogo envolve a escolha de SGBD para armazenar dados. Jogos multijogador massivo online (MMO), por exemplo, requerem bases de dados que suportam disponibilidade em larga escala, consistência e respostas em tempo real (KURABAYASHI, 2016).

Logo, partindo do pressuposto que diferentes tipos Jogos digitais precisam de níveis diferentes de quantidade de armazenamento e eficiência, tem-se por objetivo, no presente trabalho, quantificar e analisar essas diferenças. Com isso, pretende-se contribuir para a ampliação do conhecimento sobre os diferentes SGBDs presentes no mercado, qualificando os Desenvolvedores de jogos sobre esses sistemas e sobre as melhores escolhas para cada modalidade de jogo. O objetivo deste trabalho, portanto, é realizar uma análise crítica e comparativa entre dois SGBDs diferentes, a fim de especificar as particularidades de cada um deles e suas melhores aplicações. A partir disso, o trabalho mostrará qual tipo de SGBD deve ser usado para qual modalidade de jogo, ou qual finalidade é a melhor para qual SGDB, considerando a facilitação do processo de escolha de bancos de dados no princípio do desenvolvimento de jogos digitais. Por exemplo, um jogo que precisa apenas guardar a pontuação de um jogador pode fazer o uso de um SGBD que vem com ferramentas de gerenciamento de *backend*, como o Google Firebase[1]. Já jogos *multiplayer* mais complexos costumam fazer o uso

[1] https://firebase.google.com/

de um bancos de dados que tenham uma maior eficiência para atender todos os pedidos dos jogadores, como o MySQL[2], como é visto na seção de trabalhos relacionados.

Os SGBDs escolhidos para essa comparação são o MongoDB (MongoDB Inc., 2009) e o PostgreSQL (STONEBRAKER, 1996). São dois SGBDs modernos e bem consolidados, que seguem modelos diferentes de bancos de dados, então devem resultar em uma comparação relevante.

A fim de alcançar esse objetivo, pretende-se efetuar uma pesquisa preliminar sobre quais SGBDs são comumente usados em aplicações de jogos, esclarecendo suas características e propriedades. Em um segundo momento, criar-se-á um aplicação que simula as chamadas de leitura e escrita do SGBD feitas pelos jogadores. Essa aplicação será usada para mandar pedidos de leitura e escrita idênticos para dois bancos de dados, um rodando em MongoDB e outro em PostgreSQL, para que seus desempenhos possam ser comparados. Pretende-se tomar como referência a base de dados do jogo Pokémon, visto que alguns de seus dados são disponibilizados por fãs e passíveis de consulta e análise. O mesmo não acontece com a maioria dos outros jogos, pois os dados não são de conhecimento público.

O trabalho será analisado utilizando métricas de análise de desempenho comumente utilizadas na área de banco de dados. Acredita-se que se o estudo resultar em uma comparação clara de SGBDs para jogos diferentes, ele cumprirá seu objetivo, auxiliando que desenvolvedores de jogos futuros possam fazer escolhas informadas em relação ao sistema de gerenciamento de banco de dados, sendo uma contribuição para a indústria de jogos.

O restante do texto está organizado na seguinte forma. O Capítulo 2 explica a teoria por trás dos modelos dos SGBDs a serem comparados. O Capítulo 3 mostra implementações anteriores de SGDBs para uso em jogos digitais, sendo finalizado com uma análise comparativa. O Capítulo 4 ilustra a modelagem da base de dados de teste para os dois Sistemas sendo comparados. No Capítulo 5 é mostrada a implementação dos dois SGBDs e a descrição do ambiente computacional. O Capítulo 6 apresenta as conclusões e perspectivas para trabalhos futuros.

[2]https://www.mysql.com/

2 FUNDAMENTAÇÃO TEÓRICA

Nesse capítulo são apresentados os principais modelos de SGBD, junto com suas principais características.

2.1 Primeiros modelos de SGDB

Quando programas de computador começaram a ser desenvolvidos, eles focavam em cálculos e linguagens, com o computador agindo como uma calculadora, sem armazenar dados. O conceito de uma base de dados foi possibilitado, mais tarde, com a criação de tecnologias de armazenamento de dados, como discos magnéticos. Por isso, os primeiros SGDBs salvavam seus dados em uma ordem sequencial, na medida em que era o único recurso possível nesses discos.

Com o advento de discos rígidos, as possibilidades para sistemas de bancos de dados mudaram drasticamente. Os dados agora podiam ser acessados em milissegundos, independentemente da posição dos dados no disco, desse modo, os dados não precisavam mais ser organizados sequencialmente. Com isso, modelos de SGBD ficaram livres para organizar dados em estruturas mais complexas.

O primeiro modelo de SGDB a apresentar uma organização de dados é o modelo Hierárquico (TSICHRITZIS; LOCHOVSKY, 1976). Nele, dados são organizados na forma de uma árvore, na qual há um nodo raiz que armazena dados e ponteiros para outros nodos. Esses outros nodos também possuem dados e podem possuir ponteiros para outros nodos. Esse modelo foi muito útil, visto que várias situações na vida real podem ser representadas utilizando essa relação um-para-muitos. Entretanto, o modelo foi perdendo popularidade por ser rígido e não conseguir representar dados em estruturas diferentes da sua. Outro exemplo dos precursores é o modelo em rede, também conhecido como CODASYL (TAYLOR; FRANK, 1976). Como o modelo hierárquico, ele é baseado em uma estrutura de dados conhecida, sendo ela o grafo. Ela é organizada em nodos, que contém dados e links para a navegação de um nodo para outro. Esses links não precisam obedecer uma estrutura entre os nodos, fazendo com que esse modelo de SGDB seja mais

flexível que o modelo hierárquico.

2.2 Modelo Relacional

Na década de 1960, o cientista da computação Edgar Codd estava trabalhando com os modelos de SGDB detalhados anteriormente e notou as suas limitações (CODD, 1970). Esses modelos eram difíceis de serem usados por pessoas sem experiência. Também faltava uma fundamentação teórica por trás dessas bases de dados para que houvesse consistência nos dados. Por fim, as bases da época misturavam implementações lógicas e físicas, fazendo com que dados tivessem que ser salvos em formatos próximos ao físico, em vez de serem representados de uma maneira que pudesse ser entendida por um usuário sem conhecimento técnico. Codd então, em seu artigo de 1970, detalhou um modelo de SGBD que resolveria esses problemas, e que segue sendo usado até hoje.

O Modelo Relacional foi criado a partir do conceito matemático de relação, por isso o nome. Uma relação é um grupo de tuplas

$$(d_1, d_2, d_3..., d_n)$$

no qual cada elemento dj é um valor. A maneira mais comum de visualizar esse modelo é com tabelas. Normalmente, uma linha em uma tabela representa uma relação entre os elementos em um grupo de valores. Considerando que uma tabela é uma coleção dessas linhas, pode-se relacionar tabelas com o conceito de relação, com cada tupla representando uma linha na tabela. Um atributo, que corresponde a uma coluna na tabela, é a coleção de todos os valores de um tipo dj, onde j é o índice dos elementos do atributo.

Considere a Tabela 2.1, por exemplo. A Tabela apresenta 4 colunas, com nomes *ID*, *name*, *dept_name*, *salary*. Cada linha da tabela descreve um professor, com um ID, nome, nome de departamento e salário. Cada coluna da tabela, portanto, representa um atributo dos professores.

Na Tabela 2.2, estão guardados dados sobre salas de aulas, com o número de identificação da sala e o número de assentos nela. A Tabela 2.3 indica quais professores estão alocados para dar aula em quais salas,

estabelecendo uma relação entre professores e salas de aula. Ainda apresentando o conceito de relação, podemos dizer que a Tabela 2.1 relaciona o ID de cada professor com o seu nome, departamento e salário.

Tabela 2.1: Exemplo de Tabela para a relação de Professor

ID	*name*	*dept_name*	*salary*
10101	João	Ciência da Computação	65000
12121	Alice	Matemática	90000
15151	Roberto	Música	40000
22222	Pedro	Física	95000
32343	Miguel	História	60000

Fonte: Elaborada pelo autor.

Tabela 2.2: Exemplo de Tabela para a relação de sala de aula

ID_classroom	*seats*
0001	30
0002	35
0003	25
0004	32
0005	40
0006	42

Fonte: Elaborada pelo autor.

No entanto, esse modelo ainda precisava de uma maneira para garantir a validade dos dados armazenados, mesmo com a presença de erros e falhas. Se um banco usar um SGBD relacional, por exemplo, suas operações de transferência de dinheiro precisarão ser atômicas. Uma queda de energia não pode resultar que seja debitado de uma pessoa o dinheiro de uma transferência sem que a outra receba o dinheiro transferido. Isso motivou a criação de transações ACID (GRAY, 1981). ACID é um grupo de propriedades de transações de bancos de dados que tenta garantir que o SGBD não apresente erros em seus dados armazenados. As propriedades ACID são:

- **Atomicidade**. Se uma transação envolver operações menores, elas devem ser todas executadas ou nenhuma deve ser executada. Portanto, o SGBD deve possuir uma maneira de detectar se uma transação foi interrompida e retornar ao estado antes do começo dela.

Tabela 2.3: Exemplo de Tabela para a relação de alocação sala de aula

ID_classroom	*ID_teacher*
0001	10101
0002	12121
0003	12121
0004	15151
0005	22222
0006	32343

Fonte: Elaborada pelo autor.

- **Consistência**. Todas as transações possíveis de um banco de dados devem deixá-lo em um estado válido, de acordo com as regras internas da base.
- **Isolamento**. Considerando que transações devem ser unitárias, as operações internas de uma transação devem afetar o banco como se todas as outras transações não foram iniciadas ou foram completas, ainda que, na realidade, mais de uma transação esteja ocorrendo em paralelo.
- **Durabilidade**. Se uma transação é completa, os efeitos dela permanecem na base de dados, mesmo se houver uma falha no sistema.

Com a implementação dessas propriedade, SGDBs relacionais passaram a ser os mais usados no mercado, pela eficiência e robustez que podem oferecer.

A arquitetura que irá representar SGBDs relacionais será o PostgreSQL, por ser um sistema moderno e popular de banco de dados relacional.

2.3 PostgreSQL

PostgreSQL (STONEBRAKER, 1996) é um SGBD relacional com código aberto, que é considerado confiável e robusto. Por ter mais de 30 anos de desenvolvimento, ele tem vários recursos, como suporte a consultas complexas, por exemplo. O SGBD possui suporte à chaves estrangeiras, que são atributos de uma tabela que a relaciona com outra. Tem suporte a Gatilhos, que são pedaços de código procedural que são executados em

resposta a certos eventos. Também possui Visões, que são consultas à base de dados, guardadas em memória, a fim de evitar operações de consultas repetidas. No PostgreSQL, essas Visões podem ser atualizadas. As transações do Sistema também têm sua integridade preservada. Tem ainda suporte a Controle de Concorrência Multiversão, que é um método de controle de concorrência de transações. (The PostgreSQL Global Development Group, 2009). Também segue o princípio de transações ACID.

2.4 NoSQL

NoSQL é um termo genérico para denominar modelos de SGBD que não seguem o modelo relacional de bancos de dados (AMAZON, 2015). Esse modelo foi criado para ajudar empresas cujas necessidades não estavam sendo atendidas pelos SGBDs relacionais criados anteriormente. Bases NoSQL costumam ter representações dos dados mais flexíveis e, por isso, serem mais facilmente implementadas.

Modelos relacionais também têm problemas com escalabilidade. Eles não podem facilmente aumentar a eficiência de um banco de dados adicionando mais máquinas para a criação de *clusters* (LEAVITT, 2010).

Visto que bancos NoSQL oferecem maior flexibilidade que bancos relacionais, eles não costumam seguir o princípio de transações ACID. Em vez disso, eles seguem o princípio CAP (BREWER, 2000). Esse princípio diz que sistemas distribuídos devem seguir duas das três garantias a seguir:

- **Consistência**. Cada operação de leitura retorna o valor mais recente, ou um erro.
- **Disponibilidade**. Cada operação de leitura retorna dados, mas pode não ser a versão mais atualizada dos dados.
- **Tolerância de partições**. O sistema continua funcionando mesmo após falhas no sistema.

Essa escolha de garantia implica no que deve ser feito quando há uma falha em uma partição do banco de dados distribuído. Se o sistema apresentar os dados desatualizados ao usuário, a Consistência do sistema diminui para aumentar a sua Disponibilidade. Se o sistema cancelar a transação no caso de uma dessas

falhas, a Consistência aumenta e a Disponibilidade diminui.

2.5 MongoDB

MongoDB é um SGDB orientado a documentos. Esse modelo de banco de dados armazena suas informações como documentos, os quais são representados internamente como grupos de objetos, que possuem chaves distintas. Essas chaves são usadas para identificar e recuperar objetos específicos dentro de um documento. A Figura 2.1 mostra a representação de um documento em MongoDB.

Figura 2.1: Documento em uma base de dados em MongoDB.

Fonte: MongoDB Inc.

O MongoDB pode ser configurado para alcançar 2 propriedades ACID de acordo com os requisitos do usuário. Pode ter Disponibilidade e Tolerância a Partições com o uso de Replica Sets (MongoDB Inc., 2022b), e pode ter Consistência se forem aplicados os levels de Concern corretos (MongoDB Inc., 2022a).

Uma *feature* importante para o MongoDB é o *Aggregation Pipeline*[1], que é um comando que deixa o usuário agrupar dados de coleções diferentes e depois filtrá-los e ordená-los como quiser. O usuário pode programar as etapas desse comando usando "estágios" diferentes, especificados na documentação.

[1] https://www.mongodb.com/docs/manual/core/aggregation-pipeline/

3 TRABALHOS RELACIONADOS

Neste Capítulo, são apresentadas algumas soluções já realizadas para o armazenamento de dados para jogos digitais de tipos e plataformas diferentes. As soluções estão separadas em seções que agrupam jogos de tipos semelhantes. No final, é apresentada uma comparação entre as implementações dessas soluções, analisando suas especificidades e aplicações.

3.1 Jogos digitais teoria

Jogos digitais são aplicações digitais que rodam em dispositivos eletrônicos variados, com o objetivo de entreter o usuário do jogo, também chamado de jogador. É comum que jogos tenham uma conexão com um servidor, e que esse servidor tenha uma base de dados. Em 2015, já haviam 49,1 milhões de jogadores de jogos que usam conexão entre jogadores ou servidores nos Estados Unidos (Statista Research Department,, 2016). Nos dois casos a seguir, podemos ver que jogos on-line fazem uso de bancos de dados (WEILBACHER, 2012) e (GEE; KRISHNAMURTHY, 2019). Empresas de jogos digitais não disponibilizam dados da implementação de seus *backends* para o público, então não há estudos que mostram a porcentagem exata de jogos que usam bancos de dados. A seguir, são apresentadas as definições dos termos utilizados no presente capítulo relacionados aos jogos digitais .

- ***Gameplay***. *Gameplay* é um termo em inglês que denomina a maneira com a qual jogadores interagem com o jogo (SALEN; ZIMMERMAN, 2003).
- **Gênero**. O gênero de um jogo classifica ele de acordo com a maneira na qual ele é jogado (APPERLEY, 2006). Essa classificação não leva em conta aspectos gráficos do jogo. Também não considera a história do jogo. Partindo do pressuposto que é necessário programar um jogo para que ele seja jogado de uma maneira específica, podemos assumir que jogos de gêneros diferentes são implementados de maneiras diferentes.

Por terem implementações diferentes, jogos de gêneros diferentes podem necessitar de SGDBs distintos.

- **Plataforma**. Denomina o dispositivo no qual o jogo roda. Podem ser celulares, consoles dedicados a jogo, ou computadores pessoais.

3.2 Jogos Digitais Aplicativos

A partir daqui são citadas as soluções de armazenamento para alguns jogos digitais comercialmente disponíveis.

3.2.1 MMORPGs

MMORPGs (*Massively Multiplayer Online Role-Playing Game* ou Jogo de Interpretação de Personagens *Online* e em Massa para Multijogadores) é um termo criado em 1997 para designar jogos on-line que suportam muitos jogadores simultâneos, chegando até a milhões (BAILEY, 2021). Há ainda um ambiente virtual, no o qual os jogadores podem interagir em tempo real, e que continua funcionando mesmo na ausência do jogador (SAFKO; BRAKE, 2009). Esse gênero também permite aos jogadores a criação de personagens.

Em razão dessas exigências, jogos desse tipo necessitam de bancos de dados eficientes, a fim de dar conta das operações de escrita geradas por operações de atualização dos dados de personagens para uma quantidade muito alta de jogadores.

3.2.1.1 World of Warcraft

O exemplo mais conhecido desse tipo de jogo é o World of Warcraft (Blizzard Entertainment, 2008), que foi lançado em 2004 e rapidamente virou o MMORPG mais popular de todos os tempos, tendo um pico de 12 milhões de inscrições em 2010. O Sistema de Gerenciamento de Banco de Dados do jogo não é de conhecimento publico, sendo preciso deduzi-lo de outras fontes, tais como as descrições de vaga de seus desenvolvedores. De acordo com as essas vagas de trabalho para os desenvolvedores *backend* do jogo (Activision Blizzard, 2022), podemos observar que ele usa uma mistura de

MySQL e Oracle DBMS, dois SGDBs relacionais populares na época do lançamento do jogo.

Diante disso, consideramos essa escolha de SGBD bem fundamentada, pois une um sistema de banco de dados eficiente como MySQL, junto com um sistema de gerenciamento de bancos de dados em nuvem, como o Oracle. Isso faz com que o sistema do jogo possa responder à alta demanda de jogadores e ainda tenha a flexibilidade de lidar com picos de tráfego. Essa escolha, porém, deve ter gerado algumas dificuldades para a criação de conteúdo novo, por ser um banco de dados relacional e, portanto, pouco flexível.

A Figura 3.1 mostra uma foto de um jogador jogando World of Warcraft. Cada personagem com um nome roxo é um jogador, com o personagem no centro sendo o controlado pelo jogador local.

Figura 3.1: Imagem do *gameplay* do World of Warcraft.

3.2.2 Jogos *mobile*

Jogos *mobile* possuem um grande mercado de consumidores, recebendo quase 50% do rendimento da indústria de jogos digitais nos Estados Unidos (CLEMENT, 2021). A maior parte dos jogos *mobile* usam bancos de dados primariamente para salvar dados de até milhões de usuários. Por isso, precisam de SGBDs mais eficientes (KURABAYASHI,

2016).

3.2.2.1 Sonic Forces (mobile)

SEGA HARDlight é uma desenvolvedora de jogos afiliada à SEGA. De acordo com um post no blog do MongoDB (DRUMGOOLE, 2018), essa empresa migrou o *backend* de seus jogos, de servidores próprios rodando MySQL para o MongoDB Atlas, que é um banco de dados em nuvem que usa o MongoDB. Entre as razões para essa essa escolha, a empresa citou uma maior escalabilidade, menor custo de manutenção e a maior facilidade de modelar um banco de dados não relacional.

Usando esse SGDB novo, foi lançado o Sonic Forces (*mobile*)(SEGA Corporation, 2018). Ele é um jogo de corrida que usa uma conexão contínua com a internet, além de salvar dados em um *leaderboard*. Utilizando essa solução, o jogo teve uma latência baixa e nenhuma interrupção de serviço (DRUMGOOLE, 2018). Por ser um SGDB não-relacional, os dados do MongoDB podem ser reestruturados com maior facilidade, sem gerar conflitos em tabelas de dados. Já que essa escolha não gerou problemas no desempenho do jogo, ela se revelou uma boa escolha.

A Figura 3.2 mostra uma corrida contra outros jogadores no Sonic Forces.

3.2.2.2 Zynga Poker

Zynga Poker (Zynga Inc., 2007) é um jogo para plataformas *mobile* e *browser*, com conexão *online*, com a qual jogadores podem fazer partidas em tempo real de Pôquer, que é um jogo de azar, de cartas. O jogo possui um ambiente social, no qual jogadores entram em um *lobby* de cassino e podem jogar em qualquer "mesa" ou entrar em uma partida com amigos. Os jogadores escolhem mesas casuais, torneios ou mesas VIP. Um *leaderbord* mostra a colocação dos jogadores.

A Figura 3.3 mostra uma imagem promocional com uma partida de Pôquer no aplicativo.

A Zynga, desenvolvedora desse jogo, é uma das mais bem-sucedidas em jogos *mobile* do mundo, tendo 30 milhões de usuários ativos em agosto de

Figura 3.2: Imagem do *gameplay* do Sonic Forces.

2017 (TAKAHASHI, 2017). A empresa usa uma mistura de SGDBs para seus jogos, incluindo MySQL e PostgreSQL (Zynga Inc., 2022). Porém, é sabido que a zinga migrou seus jogo para o *environment* do Amazon AWS, e agora usa o DynamoDB como base de dados para o Zynga Poker (AMAZON, 2016).

3.2.2.3 Fruit Ninja

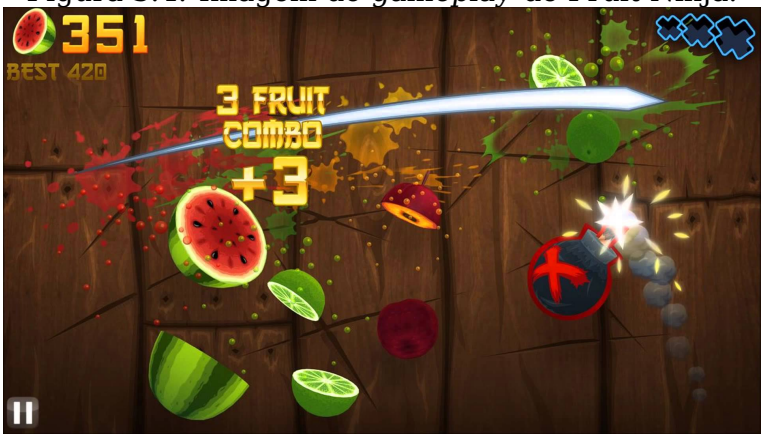
A desenvolvedora Halfbrick usa os serviços do Google Firebase como o *backend* de grande parte de seus jogos (GOOGLE, 2021). Entre esses serviços está o Realtime Database, que é um serviço de *Database As A Service* (DaaS)

Figura 3.3: Imagem promocional do *gameplay* do Zynga Poker.

usando um modelo NoSQL (GOOGLE, 2012). Junto com essa base de dados, a empresa faz uso de várias outras ferramentas do ecossistema da Google, como o Cloud Storage, outra base de dados especializada para armazenar dados como vídeos e fotos gerados pelos usuários, e o Remote Config, que permite que a aparência e comportamento do jogo seja alterado sem ser necessário um *download* de atualização do jogo (GOOGLE, 2016). Pode-se notar que a escolha de SGDB da Halfbrick se deu mais pela quantidade de ferramentas úteis para o desenvolvimento do jogo do que pelas características específicas do Firebase Realtime Database.

A maior parte dos jogo da Halfbrick são feitos para plataformas *mobile*, um dos mais populares sendo Fruit Ninja (Halfbrick Studios, 2010), que é um jogo de ação no qual o jogador tenta cortar frutas voando na tela. A Figura 3.4 mostra um momento durante o *gameplay* do Fruit Ninja

Figura 3.4: Imagem do *gameplay* de Fruit Ninja.



3.2.3 Jogos PvP

Jogos PvP (jogador contra jogador, ou player versus player) envolvem jogadores em uma partida na qual devem competir entre si. Na maior parte dos casos, eles devem se eliminar usando as mecânicas específicas do jogo. Em jogos de tiro, por exemplo, jogadores são eliminados usando armas do jogo.

3.2.3.1 Fortnite

Fortnite (Epic Games, 2017) é um dos jogos mais populares do mundo. Em um post em rede social em 2020, foi divulgado que o jogo tem mais de 350 milhões de jogadores cadastrados, que juntos jogaram por um total de 3.2 bilhões de horas (Epic Games, 2020). É um jogo multiplayer de tiro em terceira pessoa no qual os jogadores são deixados em uma ilha e devem eliminar uns aos outros com armas que encontram lá. O jogo também tem um sistema de construção com o qual jogadores podem criar estruturas para se protegerem de outros. A Figura 3.5 mostra uma captura de tela coletada durante uma partida do jogo. Para dar conta de uma base de jogadores tão grande e em um estilo de jogo no qual jogadores podem interagir entre si, a Epic Games, desenvolvedora do Fortnite, precisa de um *backend* e um SGDB extremamente eficiente.

Como discutido em um Postmortem de uma queda de serviço dos servidores de login do jogo, O MongoDB é usado para armazenar os dados de login dos jogadores (Epic Games, 2018). Além das bases de dados MongoDB, o Fortnite guarda dados usando o AWS, usando Data Lakes (AMAZON, 2021). A Amazon AWS é uma plataforma de serviços de computação em nuvem amplamente usada na indústria de jogo por efetuar o controle do hardware dos SGBDs, fazendo com que as empresas da indústria possam focar no desenvolvimento do jogo em si, sem ter que alocar seus próprios funcionários para a manutenção de bancos de dado em nuvem.

Figura 3.5: Imagem do *gameplay* de Fortnite.

3.2.4 Pokémon

Nessa seção é descrito o jogo do qual vem os dados do banco de dados de teste para a comparação entre os SGBDs. Pokémon (The Pokémon Company, 1998) é um jogo no qual o jogador deve capturar criaturas, também chamadas de *Pokémons*, e batalhar outros personagens controlados pelo jogo, ou até por outros jogadores, em batalhas em turnos. A franquia é a terceira mais bem-sucedida do mundo com 440 milhões de cópias do jogo vendidas (The Pokémon Company, 2021).

Durante uma batalha, *Pokémons* fazem ataques entre si. Ataques reduzem a os pontos de vida (também chamado de "HP") do *Pokémon* adversário, e se o HP do *Pokémon* atingido chega a zero, a criatura é derrotada. A Figura 3.6 Mostra um momento em uma batalha. Para implementar esse sistema de combate e torná-lo interessante, o jogo precisa aplicar valores que varia a quantidade de dano dado de acordo com o ataque e os *Pokémons* envolvidos. Por exemplo, *Pokémons* tem "tipos" diferentes, que melhoram ou pioram a efetividade de seus ataques.

Pokémons da mesma espécie também possuem traços que os diferenciam entre si, como o seu nível, que indica o quanto de experiência a criatura obteve batalhando. Treinadores, que são os personagens controlados por jogadores ou pelo jogo, podem capturar quantos pokémons quiserem, mas só podem levar até 6 para uma batalha. Esse ambiente, com espécies, *Pokémons* individuais e treinadores gera um domínio de dados interessante para uma análise de SGDB, por gerar informações o suficiente

para a efetuação de testes.

Figura 3.6: Imagem de uma batalha no jogo Pokémon

3.2.5 Comparação

Nessa seção é apresentada a comparação de implementações de SGBDs para aplicações em jogos digitais com a Tabela 3.1 e a Tabela 3.2, para que seja possível ver as variadas implementações bancos de dados para jogos diferentes. O item "SGBD" mostra o Sistema de Gerenciamento de Banco de Dados usado para suportar o banco de dados usado no jogo. "Plataforma" mostra quais dispositivos podem ser usados para jogar o jogo na coluna. O item "Gênero de jogo" mostra o gênero atribuído ao jogo, pelos desenvolvedores ou pelos jogadores. O item "Computação em Nuvem" mostra qual solução de Computação em nuvem é usada.

Tabela 3.1: Comparação entre os trabalhos

	World of Warcraft	**Sonic Forces**	**Zynga Poker**
SGDB	MySQL	MongoDB	DynamoDB
Plataforma	PC	*mobile*	PC e *mobile*
Gênero de jogo	MMORPG	Corrida *Online*	Cartas *Online*
Computação em Nuvem	Proprietário - Usando Oracle RDBMS	MongoDB Atlas	Amazon AWS

Fonte: Elaborada pelo autor.

Computação em Nuvem é a oferta de recursos computacionais, como poder computacional e, no caso dessas SGDBs, armazenamento de dados, sem o gerenciamento ativo do usuário. Serviços como o Amazon AWS oferecem essa computação para outras empresas por um custo fixo por mês.

Tabela 3.2: Comparação entre os trabalhos- 2

	Fruit Ninja	**Fortnite**	**Este trabalho**
SGDB	Firebase Real Time Database	MongoDB e AWS Data Lakes	MongoDB e PostgreSQL
Plataforma	*mobile*	Windows, Nintendo Switch, PlayStation 4 e 5, Android, Xbox One e Xbox Series X/S	Não se aplica
Gênero de jogo	Ação- Single Player	Multiplayer Battle Royale	Não se aplica
Computação em Nuvem	Google Firebase	Proprietários e AWS	Não usado

Fonte: Elaborada pelo autor.

Empresas de jogos, por sua vez, fazem amplo uso desses serviços, para que possam focar no desenvolvimento do jogo em si, sem ter que alocar seus próprios recursos para a manutenção de bancos de dado em nuvem. O único exemplo de uma desenvolvedora de jogos usar uma solução própria de Computação em Nuvem entre as pesquisadas é a Blizzard, desenvolvedora do World of Warcraft, que foi lançado em 2005, antes de surgirem esses serviços de armazenamento em nuvem. Já que jogos em plataformas e estilos diferentes fazem uso de SGBDs diferentes, essa comparação mostra que esses jogos têm necessidades diferentes para Sistemas de Gerenciamento de Bancos de Dados. Os SGBDs usados nesse trabalho serão instanciados localmente, para que suas comparações possam ser feitas em um ambiente controlado, então não serão testados em nuvem.

4 MAPEAMENTO DE DADOS

Neste Capítulo, são especificadas as duas abordagens de representação da base de dados de *Pokémon*, modeladas para os dois SGBDs sendo testados. As abordagens são arbitrárias, já que implementações de bases de dados para o jogo oficial não são públicas. Dados das espécies de *Pokémon* foram coletadas de um site para bases de dados (BARRADAS, 2021) e dados de treinadores e times foram tirados de uma página diferente, no mesmo site (CUSACK, 2017). Para as duas seções seguintes, são apresentados os mapeamentos dos dados para os dois SGBDs.

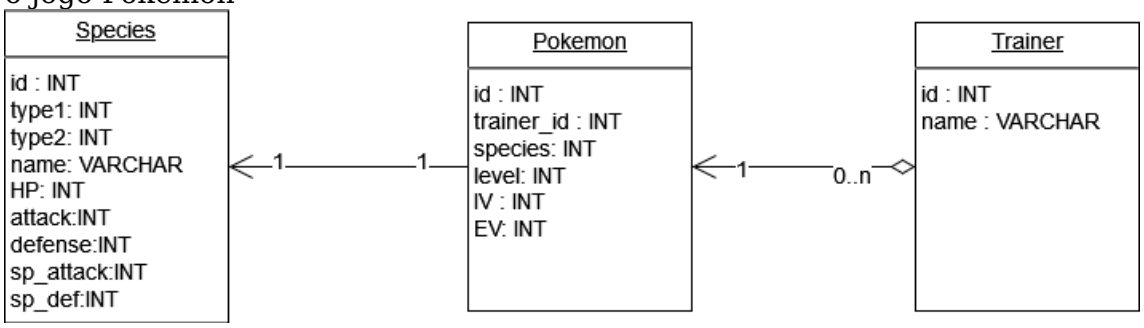
4.1 Mapeamento para o banco relacional

Nesta seção, é descrito o Mapeamento dos dados para o banco de dados relacional PostgreSQL. Para que os dados possam ser inseridos no banco de dados relacional PostgreSQL, é necessário criar um modelo de dados, para que o SGBD possa gerar as tabelas para conter e criar relações entre os dados corretamente. O diagrama entidade-relacionamento na Figura 4.1 mostra a relação entre os dados do domínio do jogo *Pokémon*. Segue a explicação das tabelas do modelo, e o que os seus atributos significam para o jogo.

- *Species*. Denomina a "espécie" do *Pokémon*. Cada espécie tem um ou dois tipos, que são propriedades que influenciam a efetividade de certos ataques em uma batalha. O valor "name" denomina o nome da espécie do *Pokémon*. Os valores "HP", "attack", "defense", "sp_attack" e "sp_def" são valores que são usados no cálculo de dano causado a um *Pokémon* (BULBAPEDIA, 2007).
- *Pokemon*. Denomina um *Pokémon* de uma espécie específica. Os atributos "level", "IV" e "EV" influenciam os atributos do personagem, para que mais de um *Pokémon* da mesma espécie possam ter atributos diferentes (BULBAPEDIA, 2005). Por não serem inclusos na fonte de dados, O "IV" e "EV" tiveram de ser omitidos para a implementação dos SGDBs.
- *Trainer*. Denomina um treinador, que possui um número arbitrário de

Pokémons, podendo ser um jogador ou um personagem controlado pelo jogo.

Figura 4.1: Diagrama entidade-relacionamento para o domínio dos dados para o jogo Pokemon

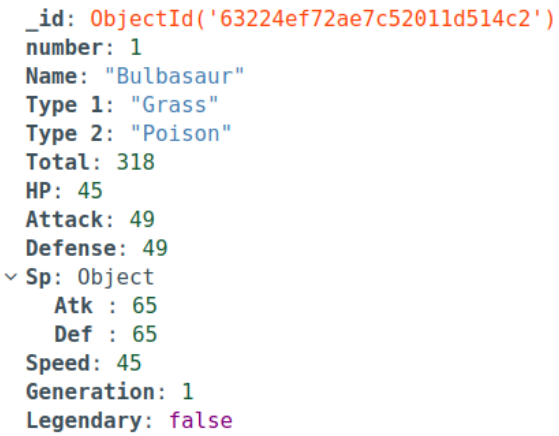


Fonte: Autor.

4.2 Mapeamento para banco de dados baseado em documentos

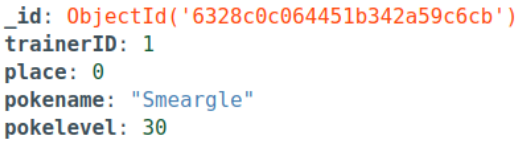
Nesta seção, é descrita a inserção dos dados no banco de dados do MongoDB. Para a inserção em um banco de dados baseado em documentos, não é necessário efetuar modelagem dos dados, já que a estrutura desse modelo é menos rígida que a do modelo relacional. Alguns dados foram removidos do arquivo CSV de *Pokémons* individuais, já que eles devem ser recuperados da coleção de dados das suas espécies. Depois disso, os arquivos CSV dos dados foram inseridos diretamente na *database* do MongoDB com o comando *mongoimport*. A Figura 4.2 Mostra o modelo gerado pelo MongoDB para armazenar os dados das espécies. A Figura 4.3 mostra a coleção gerada para armazenar dados de *Pokémons*. A Figura 4.4 mostra a coleção de dados de treinadores.

Figura 4.2: Estrutura da coleção de espécie de *Pokémon* no MongoDB



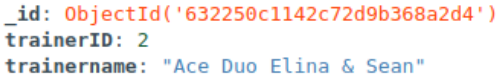
Fonte: Autor.

Figura 4.3: Estrutura da coleção de *Pokémons* específicos no MongoDB



Fonte: Autor.

Figura 4.4: Estrutura da coleção de treinadores no MongoDB



Fonte: Autor.

5 AVALIAÇÃO EXPERIMENTAL

Neste capítulo são apresentados os experimentos feitos nos dois SGBDs, junto com uma explicação do ambiente computacional. Depois são apresentadas as conclusões sobre esses experimentos, e qual sistema seria o melhor para qual tipo de jogo. Também são apresentadas as limitações dos testes.

5.1 Ambiente computacional

Os testes foram feitos em um computador com processador *AMD Ryzen 5 3600X*. Possui 16GB de memória RAM DDR4. O Sistema Operacional usado é o Ubuntu 20.04 LTS (Focal Fossa). Os pedidos de dados foram feitos usando a ferramenta Postman.

5.1.1 Implementação do banco PostgreSQL

Para a implementação do banco de dados em PostgreSQL, foi usada a linguagem Python, usando o *framework* Flask (RONACHER, 2010), que possibilita a criação rápida de aplicações conectadas com bases de dados, junto com a biblioteca SQLAlchemy, que adiciona comandos para manipular bancos de dados relacionais. Também é feito uso da ferramenta Postbird (EVSTIGNEEV, 2014), que permite a visualização das tabelas criadas.

Para criação das tabelas para o banco de dados relacional, os dados dos documentos CSV tiveram que ser alterados durante a importação. A tabela de *Pokémons* individuais tem dados que foram descartados, já que são armazenados na tabela de espécies. O *foreign key species_id*, que junta cada *Pokémon* a uma espécie, teve de ser criada comparando os nomes das criaturas nas duas tabelas. Isso gerou alguns problemas, já que a grafia de alguns *Pokemons* estava diferente entre as duas tabelas.

5.1.2 Implementação do banco MongoDB

Para o banco de dados MongoDB, também foi usado o *framework* Flask em Python, usando, neste caso, a biblioteca Flask-PyMongo para a integração com o SGDB. Foi usada a ferramenta MongoDB Atlas para a visualização das coleções inseridas.

5.2 Testes sobre os SGDBs

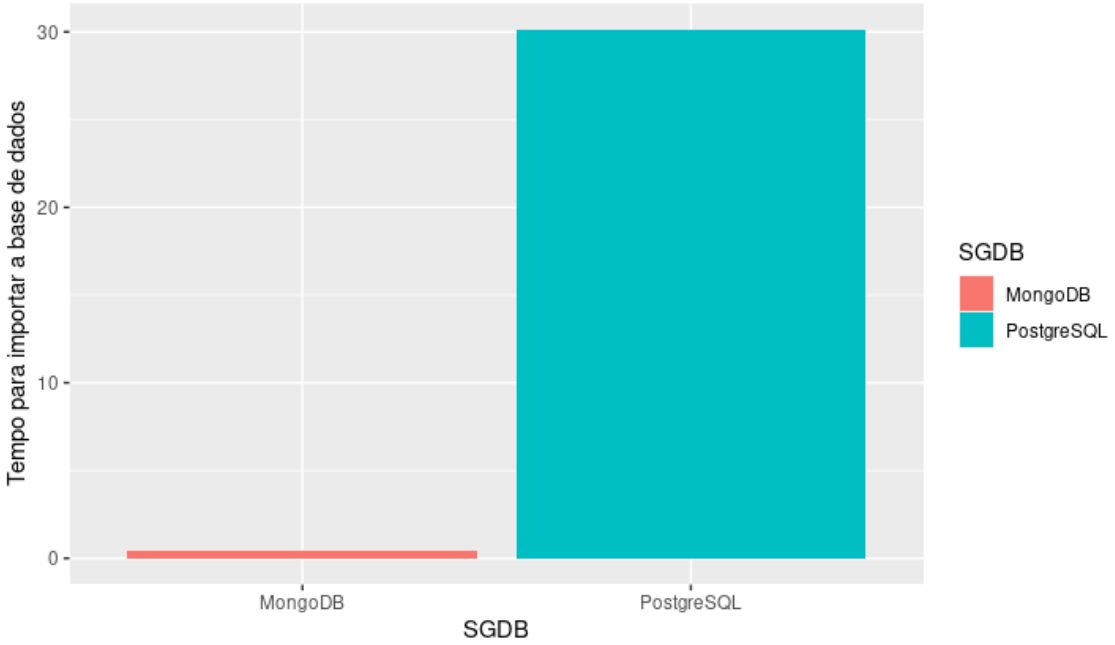
Aqui são apresentados testes sobre dos dois SGBDs. São feitas *queries* que devem retornar resultados iguais usando os dois sistemas, e é comparado o tempo de resposta. Há muita pouca literatura sobre quais pesquisas são as mais usadas na área de jogos digitais, em razão da não divulgação por parte das empresas de jogos. Por isso, serão usadas pesquisas relevantes ao jogo Pokémon, visto haver mais dados divulgados pelos usuários. Para gerar os resultados dos testes, cada consulta foi executada 10 vezes, sendo apresentada a média dessas execuções.

5.2.1 Armazenamento em disco

A carga das 3 coleções no MongoDB demorou 0,429 segundos, mais o processo de ajustar os booleanos da coleção, que demorou 0,015 segundos, enquanto que o processo de importação dos dados para o PostgreSQL demorou 29,79 segundos. Para o ajuste dos IDs dos pokémons, demorou 0,32 segundos. A Figura 5.1 ilustra a diferença no tempo de carga entre os dois sistemas.

Os dados e registros no SGBD MongoDB ocuparam 1,5MB. O comando usado para mostrar essa informação foi *db.stats()*. O banco de dados ocupa 14MB no PostgreSQL. Foi usado o comando *pg_database_size()* para recuperar esse dado. Essa diferença mostra que o MongoDB é mais econômico no uso da memória do sistema, já que as tabelas em si ocupam apenas 5,875MB. A comparação de espaço de armazenamento está na Figura 5.2

Figura 5.1: Comparação entre os tempos de execução para a carga dos dados em cada SGBD.



Fonte: Autor.

5.2.2 *Pokédex*

No jogo Pokémon, os jogadores podem acessar uma lista com todos os *Pokémons* do jogo, e ver suas característica. Essa lista se chama *Pokédex*[1]. O teste a seguir tenta verificar qual SGDB tem mais facilidade ao retornar essa lista ao jogador. Como podemos ver na Figura 5.3, o tempo de resposta do PostgreSQL foi de 0,025 segundos, e do MongoDB foi de 0,035 segundos. Essa diferença de tempo de resposta mostra que o MongoDB é mais lento para consultas mais simples. Os dois SGDBs tentaram retirar os dados de apenas uma fonte, sendo ela uma tabela para o Postgres e uma coleção para o MongoDB. A formatação dos dados da resposta está representada na Figura 5.4.

Consulta PostgreSQL

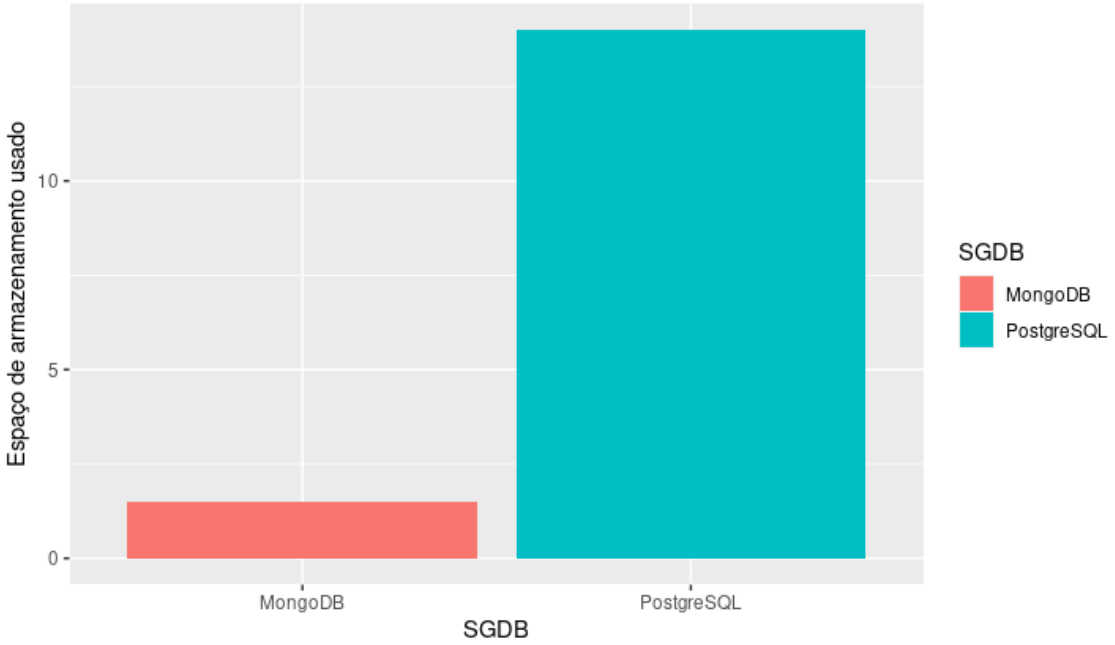
```
SELECT *
    FROM species
```

Consulta MongoDB

```
mongo.db.pokedex.find()
```

[1] https://www.pokemon.com/br/pokedex/

Figura 5.2: Comparação entre os tempos de execução para a consulta em cada SGBD.



Fonte: Autor.

5.2.3 *Pokémons* com valores associados

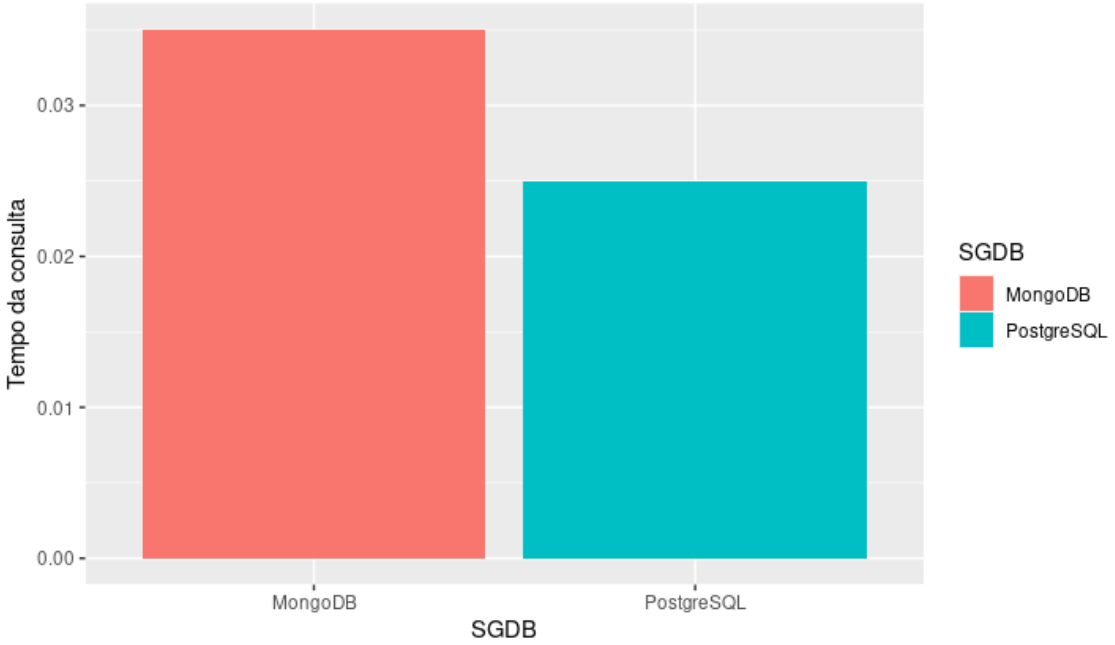
Como consta na Subseção 3.2.4, o jogo Pokémon determina dados de *Pokémons* individuais a partir de suas espécies. Portanto, é relevante haver uma *query* que retorna *Pokémons* com os dados de suas espécies.

Os dados do banco de dados MongoDB estão divididos em 3 coleções. Por isso, foi necessário usar o *Aggregate Pipeline* do SGDB. Foi usado o comando *$lookup* para juntar a coleção de espécies com a de *Pokémons* individuais, e depois o comando *$sort* para ordenar as entradas da resposta. Por causa do funcionamento interno do MongoDB, os dados da espécie de *Pokémon* estão dentro do objeto "stats", como pode ser visto na figura 5.5.

Os dados de espécies e de *Pokémons* individuais estão divididos em 2 tabelas diferentes no banco de dados PostgreSQL. Por isso, foi feito uso do comando JOIN, para unir os dados das duas tabelas, usando o *id* da espécie do *Pokémons*. Os dados retornados pela consulta SQL estão também na Figura 5.5.

Como pode ser visto na Figura 5.6, o tempo de resposta do PostgreSQL foi de 0,24 segundos na primeira execução, e por volta de 0,17 nas execuções seguintes. As queries para o banco de dados MongoDB foi de 10,89 segundos,

Figura 5.3: Comparação entre os tempos de execução para a consulta em cada SGBD.



Fonte: Autor.

sem haver uma redução no tempo de execução em execuções seguintes. Com essa comparação, podemos ver que o comando *$lookup* do MongoDB é muito menos eficiente que um JOIN do PostgreSQL.

Os trechos de código usados para as pesquisas seguem abaixo.
O código para a pesquisa para o PostgreSQL segue abaixo.

```
SELECT POK.id, POK.trainer_id, POK.place, POK.level, SP.dex,
SP.name, SP.type1, SP.type2, SP.\"HP\", SP.attack,
SP.defense, SP.sp_atk, SP.sp_def, SP.speed
FROM species SP
LEFT JOIN pokemon POK
ON SP.id = POK.species_id
ORDER BY dex ASC
```

Figura 5.4: Resultado da query de **Pokedex** nos dois SGBDs

Fonte: Autor.

O código para a pesquisa para o MongoDB segue abaixo.

```
mongo.db.pokemon.aggregate( [
    {'$lookup':
       {
         'from': "pokedex",
         'localField': "pokename",
         'foreignField': "Name",
         'as': "stats"
       }
    },
    {'$sort' : {
        'stats.number': 1
        }
    }
])
```

Figura 5.5: Resultado da query de *stats* no PostgreSQL e MongoDB

Fonte: Autor.

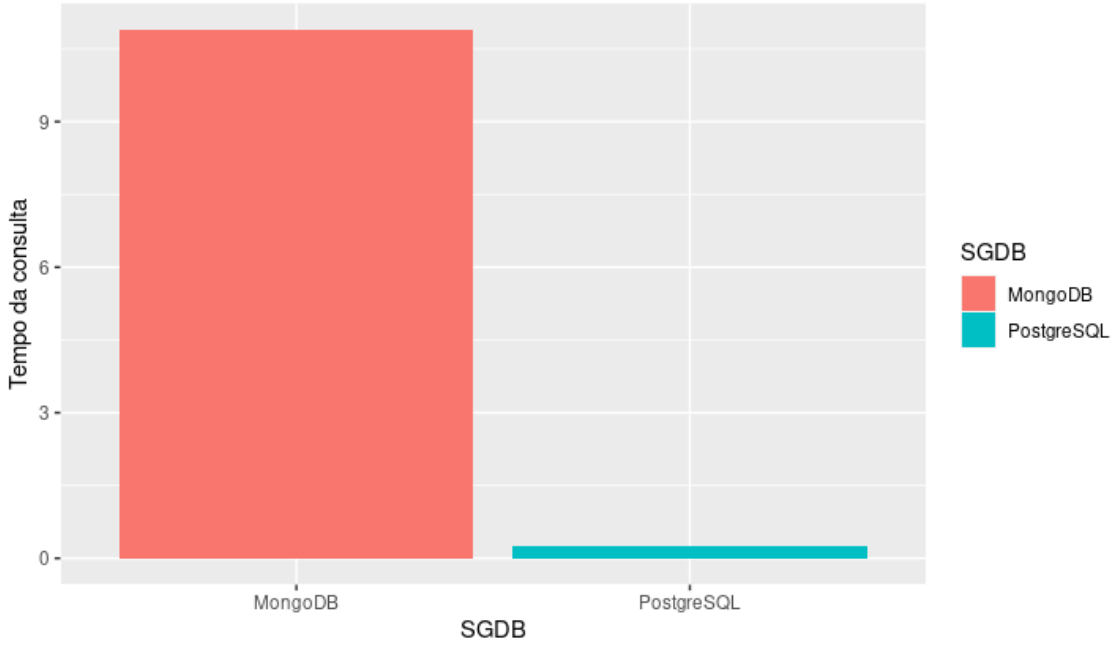
5.2.4 *Pokémons* Lendários

"Lendário" é uma atribuição que se dá a *Pokémons* que são mais raros e mais difíceis de serem capturados que outros. A consulta a seguir retorna as espécies de *Pokémon* que são lendárias, como pode ser visto na Figura 5.10. A Figura 5.8 mostra o tempo de resposta das consultas nos dois SGDBs. O banco de dados em PostgreSQL demorou 0,038 segundos na primeira consulta e MongoDB demorou 0,009 na primeira. Nas consultas seguintes, os tempos de resposta dos dois SGDBs diminuiram, por causa do caching dos bancos de dados. O PostgreSQL demorou 0,0016 segundos e MongoDB demorou 0,0051 segundos. A Figura 5.9 mostra a comparação dos tempos depois da primeira consulta.

O Flask SQLAlchemy não usa caching nativamente[2], e a aplicação gera uma nova conexão com o PostgreSQL a cada execução, não guardando cache[3].

[2]https://docs.sqlalchemy.org/en/14/orm/examples.html#module-examples.dogpile_caching
[3]https://docs.sqlalchemy.org/en/14/core/connections.html#connectionless-execution-implicit-execution

Figura 5.6: Comparação entre os tempos de execução para a consulta em cada SGBD.



Fonte: Autor.

Por isso, o "caching" notado anteriormente vem do Sistema Operacional no qual o SGDB está rodando, então, para limpar a cache, a aplicação deve ser resetada.

Pode-se ver que o MongoDB tem vantagem nessa consulta, já que ela envolve apenas uma coleção. Por causa do *caching* do PostgreSQL, o SGDB volta a ser mais eficiente que o MongoDB após a primeira consulta.
O código para a pesquisa para o PostgreSQL segue abaixo.

```
SELECT *
    FROM species
    WHERE legendary = true
```

O código para a pesquisa para o MongoDB segue abaixo.

```
mongo.db.pokedex.find({"Legendary" : True})
```

5.2.5 Times

Um dos objetivos do jogo é capturar e colecionar *Pokémons*. Por isso, é importante para o jogo poder saber quais pertencem a quais treinadores. A

Figura 5.7: Resultado da query de lendários no PostgreSQL e MongoDB

```
[
    {
        "id": "157",
        "dex": "144",
        "name": "Articuno",
        "type1": "Ice",
        "type2": "Flying",
        "HP": 90,
        "attack": 85,
        "defense": 100,
        "sp_atk": 95,
        "sp_def": 125,
        "speed": 85,
        "gen": 1,
        "legendary": true
    },
```

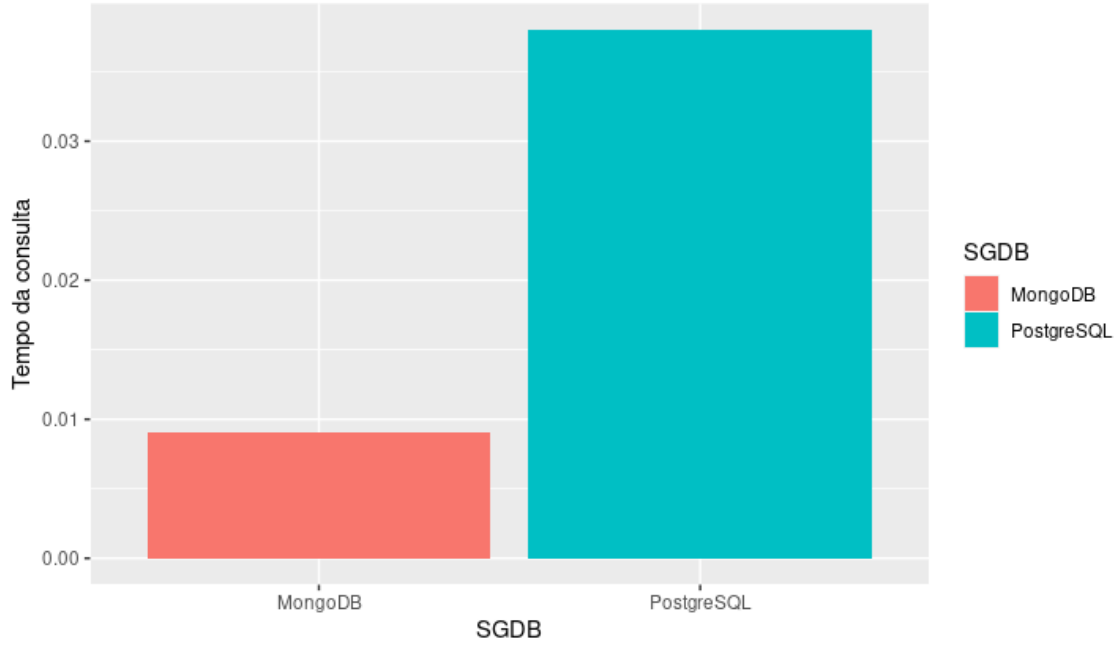
```
[
    {
        "_id": {
            "$oid": "63224ef72ae7c52011d5155e"
        },
        "number": 145,
        "Name": "Zapdos",
        "Type 1": "Electric",
        "Type 2": "Flying",
        "Total": 580,
        "HP": 90,
        "Attack": 90,
        "Defense": 85,
        "Sp": {
            " Atk": 125,
            " Def": 90
        },
        "Speed": 100,
        "Generation": 1,
        "Legendary": true
    },
```


Fonte: Autor.

consulta a seguir retorna todos os treinadores, junto com seus *Pokémons*. Os resultados das consultas estão ilustrados na Figura 5.10 A consulta em PostgreSQL demorou 0,16 segundos, enquanto que a em MongoDB demorou 117,127 segundos. Isso mostra que a operação de junção de coleções diferentes do MongoDB é menos eficiente que uma junção de tabelas do PostgreSQL.
O código para a pesquisa para o PostgreSQL segue abaixo.

```
SELECT T.id, T.name, P.id, P.species_id, P.place, P.level
    FROM trainer T
    LEFT JOIN pokemon P
    ON T.id = P.trainer_id
    ORDER BY T.id ASC
```

Figura 5.8: Comparação entre os tempos de execução para a consulta em cada SGBD.



Fonte: Autor.

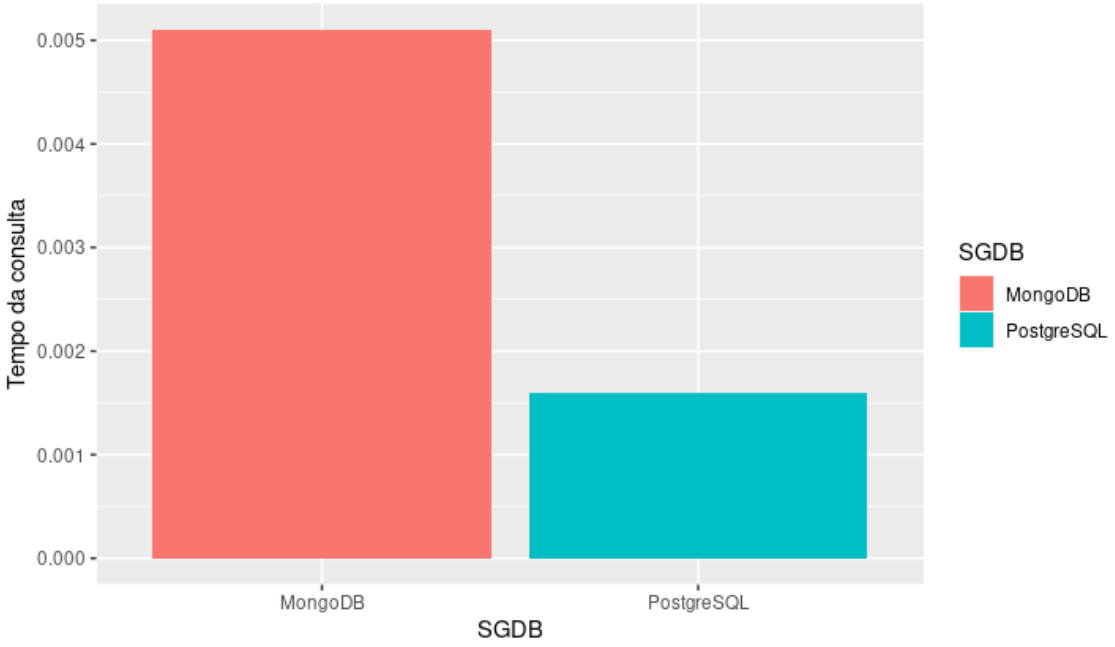
O código para a pesquisa para o MongoDB segue abaixo.

```
mongo.db.trainers.aggregate( [
    {'$lookup':
        {
          'from': "pokemon",
          'localField': "trainerID",
          'foreignField': "trainerID",
          'as': "pokemon"
        }
    }
])
```

5.2.6 Filtro de *Pokémons* por level

O *level* de um *Pokémon* é importante para determinar o quão forte ele é. Por isso, faz sentido haver uma consulta que retorna uma lista de *Pokémons* que têm um *level* acima de uma constante. No PostgreSQL, o tempo de resposta foi em média 0,0616 segundos na primeira consulta e

Figura 5.9: Comparação entre os tempos de execução para a consulta em cada SGBD.



Fonte: Autor.

0,039 nas seguintes. A consulta MongoDB demorou 0,22 segundos. A comparação está ilustrada na Figura 5.11. Como podemos ver, o MongoDB é menos eficiente para consultas com filtragem de dados.
O código para a pesquisa para o PostgreSQL segue abaixo.

```
SELECT *
    FROM pokemon
    WHERE level > 40
```

O código para a pesquisa para o MongoDB segue abaixo.

```
mongo.db.pokemon.find({"pokelevel" : { "$gt" : 40 }})
```

5.2.7 Limitações

A comparação foi mais detalhada com testes de leitura e agrupamento de dados, e menos com alteração e inserção de dados. Além disso, no banco de dados MongoDB, os dados foram inseridos sem muitas alterações. Porém, juntar as três coleções em uma única colecão poderia ter melhorado a eficiência das consultas que envolvem o comando *$lookup*, já que é dito que o

Figura 5.10: Resultado da query de times no PostgreSQL e MongoDB

Fonte: Autor.

MongoDB lida melhor com filtragens dentro de coleções do que junções de coleções diferentes.

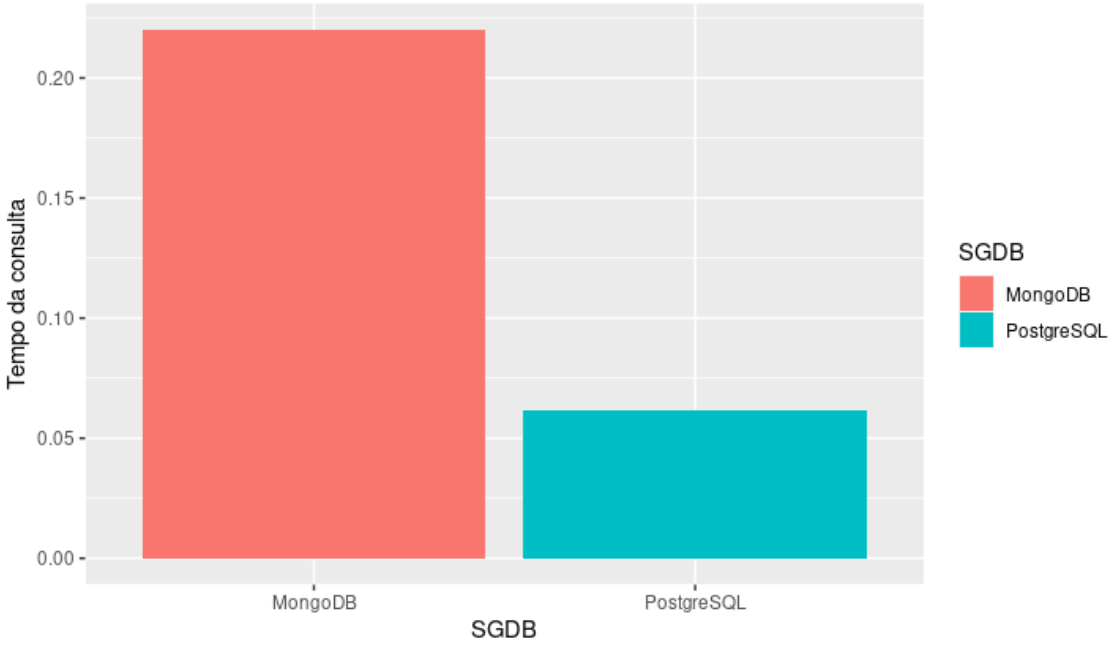
5.3 Análise Geral dos Resultados

O banco de dados em MongoDB teve um melhor desempenho na consulta de Pokémons lendários, ao que tudo indica, em razão de ser feito apenas um teste, em um atributo, em uma coleção, mostrando que o SGBD possui uma maior eficiência com filtragens simples. O banco de dados também obteve um desempenho muito melhor na importação de arquivos, pois os seus dados não precisaram passar por uma etapa de processamento de informações, como aconteceu com o PostgreSQL.

O PostgreSQL, por sua vez, teve um melhor desempenho juntando dados de tabelas diferentes, assim como com a maior parte das operações de filtragem de entradas.

A partir do dito até aqui, recomenda-se usar o MongoDB para guardar dados dos usuários. Pois, além de ter os benefícios de poder armazenar dados não-estruturados - por ser um banco de dados orientado a documentos

Figura 5.11: Comparação entre os tempos de execução para a consulta em cada SGBD.



Fonte: Autor.

- ele pode guardar qualquer tipo de dado que os desenvolvedores extraírem ao monitorar seus jogadores. Ele também parece ter uma maior eficiência ao adicionar dados novos à coleções, então pode lidar com uma quantidade elevada de dados provindos de muitos jogadores. Além dos testes efetuados neste trabalho, as implementações anteriores também confirmam essa hipótese. MMORPGS, como dito na Subseção 3.2.1, fazem muitas operações de escrita para dados de muitos jogadores. Também foi demonstrado que jogos *mobile* precisam de bancos de dados para salvar dados de até milhões de jogadores. O jogo Sonic Forces usa o MongoDB justamente para isso. O Zynga Poker e o Fruit Ninja usam serviços de Database As A Service para guardar dados de usuários, sendo que esses serviços usam modelos NoSQL, que o MongoDB também segue. O jogo Fortnite, como dito na Subseção 3.2.3.1, também faz uso do MongoDB, para salvar dados dos cadastros dos seus jogadores.

Quanto ao PostgreSQL, considera-se ser um banco de dados recomendado para armazenar dados requisitados várias vezes, por vários usuários. Isso indica que PostgreSQL deve ser usado na lógica interna de servidores de jogos. Como pode ser visto em (WEILBACHER, 2012), bancos de dados relacionais são usados para organizar partidas pela internet entre

jogadores diferentes, guardando dados das contas e dispositivos dos jogadores na partida. Em casos como este, os dados de tabelas diferentes precisam ser unidos eficientemente para que o servidor possa reunir os dados dos participantes da partida e iniciá-la em um tempo hábil. O PostgreSQL pode, portanto, desempenhar o papel de SGBD neste caso, por conseguir reunir e filtrar dados de grupos diferentes mais rapidamente que o MongoDB. Também é dito na apresentação citada que o tamanho do banco de dados não precisa aumentar depois de adicionar todas as contas de usuários, então a falta de eficiência na carga de dados vista antes não é um problema neste caso.

6 CONCLUSÃO

Neste capítulo são apresentadas as conclusões sobre o trabalho realizado.

6.1 O que Foi Apresentado

Sob o entendimento que bancos de dados são empregados em grande parte dos jogos digitais hoje em dia, este trabalho apresentou uma comparação de SGBDs para desenvolvimento de jogos digitais, com o objetivo de ajudar desenvolvedores na escolha de bancos de dados. Para esta comparação, foram mostrados os principais modelos de SGBD, junto com um histórico de modelos usados no passado. Foram apresentados os modelos Hierárquico e CODASYL, junto com os dois modelos usados pelos SGBDs na comparação: o modelo Relacional e o modelo NoSQL. Foi abordado o princípio de Transações ACID em bancos de dados relacionais, e o princípio CAP, para bancos de dados NoSQL. Em seguida, foram descritos os SGDBs e suas ferramentas relevantes ao trabalho.

Depois, foram apresentados alguns termos usados para descrever jogos digitais, para que logo fossem apresentadas algumas implementações existentes de bancos de dados para os mesmos, assim como as descrições desses mesmos jogos. Feito isso, foi explicado o jogo *Pokémon*, visto que o *dataset* usado nos testes provêm desse jogo.

Com isso feito, foi descrito o mapeamento e a modelagem dos dados usados para os testes. Seguindo isso, foi descrito o ambiente computacional e a implementação dos SGBDs. Por fim, foram descritas as avaliações e conclusões sobre o MongoDB e o PostgreSQL.

6.2 Contribuições

O trabalho demostrou as diferenças entre os SGDBs, como a modelagem dos dados para a inserção nos bancos, e como é mais complexo modelar dados para SGDBs relacionais. Também apontou as diferenças nos

dois SGDBs na etapa de inserção de dados, e como o MongoDB é superior nesse quesito por não precisar alterar os dados antes de inseri-los. O MongoDB também pareceu mais econômico com a quantidade de memória usada para guardar os dados. Operações de filtragens de dados foram mais eficientes no MongoDB quando a operação de filtragem é simples, porém o PostgreSQL é vantajoso com comparações um pouco mais complexas, como testes de "maior". O PostgreSQL foi mais eficiente ao retornar as tabelas de um tipo que o MongoDB ao retornar todas as entradas em uma coleção. Os dados do MongoDB foram separados em coleções diferentes da mesma maneira que foram separados em 3 tabelas diferentes no PostgreSQL, e este último teve uma eficiência muito melhor ao unir esses dados das diferentes coleções. Com esses resultados catalogados, esse TCC deve servir para auxiliar na escolha de banco de dados para o desenvolvimento de jogos digitais.

6.3 Trabalhos Futuros

Para trabalhos futuros, poderia haver um aprofundamento nos testes realizados, a fim de gerar recomendações mais acuradas. Poderiam ser feitos, por exemplo, mais testes de escrita, avaliando a capacidade dos SGDBs de guardar rapidamente os dados. Para os testes no MongoDB, juntar as coleções em uma só poderia melhorar a eficiência das consultas que envolvem o comando *$lookup*, na medida em que é dito que o MongoDB lida melhor com filtragens dentro de coleções do que junções de coleções diferentes.

Os testes deste trabalho foram feitos com o *core* base do MongoDB e do PostgreSQL. Trabalhos futuros poderiam fazer uso de técnicas como Índices, para aumentar a performance do SGBD ao máximo. [1] [2]

Outra possibilidade para trabalhos futuros seria variar o ambiente do *back-end* dos bancos de dados. O *framework Flask* usado no trabalho é útil para a prototipação e teste de bancos de dados, mas ele é feito exclusivamente para a linguagem Python, que é considerada mais lenta que

[1] https://www.postgresql.org/docs/current/indexes.html
[2] https://www.mongodb.com/docs/manual/indexes/

outras (GeeksForGeeks, 2021). Isso pode ter efeitos na comparação de tempo de carga de dados, pois a importação de dados para o PostgreSQL foi manejada por essa biblioteca e por código autoral em Python, enquanto que o MongoDB não precisou fazer uso disso. Portanto, talvez uma mudança de linguagem leve a um aumento na eficiência da carga dos dados.

Para prosseguir este trabalho, os SGDBs também poderiam, ainda, ser testados na presença de falhas, para que fosse possível ver seus efeitos na consistência de dados ou na disponibilidade do sistema. Isso contribuiria para demostrar a diferença entre as transações ACID do PostgreSQL e o princípio CAP do MongoDB.

Além de falhas no sistema, também poderiam ser feitos testes com simulações de problemas de rede, como alta latência, também chamada de *lag*. Esse problema é relevante porque grande parte dos jogadores de jogos on-line se frustram e param de jogar jogos com alta latência (Edgegap, 2022).

REFERÊNCIAS

Activision Blizzard. **Job Openings at Blizzard**. 2022. Acesso em: 25/08/2022. Disponível na Internet: <https://careers.blizzard.com/global/en>.

AMAZON. **O que é NoSQL?** 2015. Disponível na Internet: <https://aws.amazon.com/pt/nosql/>.

AMAZON. **Zynga Case Study**. 2016. Acesso em: 25/08/2022. Disponível na Internet: <https://aws.amazon.com/pt/solutions/case-studies/zynga/>.

AMAZON. **Epic Games on AWS**. 2021. Acesso em: 25/08/2022. Disponível na Internet: <https://aws.amazon.com/pt/solutions/case-studies/innovators/epic-games/>.

APPERLEY, T. H. Genre and game studies: Toward a critical approach to video game genres. **Simulation & Gaming**, v. 37, n. 1, p. 6–23, 2006. Acesso em: 12/09/2022. Disponível na Internet: <https://doi.org/10.1177/1046878105282278>.

BAILEY, D. **Blizzard's player count is unchanged this quarter, despite controversy and Diablo 2 launch**. 2021. Acesso em: 25/08/2022. Disponível na Internet: <https://www.pcgamesn.com/world-of-warcraft/blizzard-player-count-2021>.

BARRADAS, A. **Pokemon with stats**. 2021. Acesso em: 25/08/2022. Disponível na Internet: <https://www.kaggle.com/datasets/abcsds/pokemon>.

Blizzard Entertainment. **World of Warcraft**. 2008. Acesso em: 25/08/2022. Disponível na Internet: <https://worldofwarcraft.com>.

BREWER, E. Towards robust distributed systems. Em: . [S.l.: s.n.], 2000. p. 7.

BULBAPEDIA. **Stat**. 2005. Acesso em: 25/08/2022. Disponível na Internet: <https://bulbapedia.bulbagarden.net/wiki/Stat>.

BULBAPEDIA. **Damage**. 2007. Acesso em: 25/08/2022. Disponível na Internet: <https://bulbapedia.bulbagarden.net/wiki/Damage#Damage_calculation>.

CLEMENT, J. **Mobile gaming market in the United States - statistics & facts**. 2021. Acesso em: 25/08/2022. Disponível na Internet: <https://www.statista.com/topics/1906/mobile-gaming/#topicHeader__wrapper>.

CODD, E. F. A relational model of data for large shared data banks. **Comm. ACM**, Science and Information Organization, v. 13, n. 6, p. 377–387, 1970.

CUSACK, L. **Pokemon Trainers Dataset**. 2017. Acesso em: 25/08/2022. Disponível na Internet: <https://www.kaggle.com/datasets/lrcusack/pokemontrainers>.

DRUMGOOLE, J. **SEGA HARDlight Migrates to MongoDB Atlas to Simplify Ops and Improve Experience for Millions of Mobile**. 2018. Acesso em: 25/08/2022. Disponível na Internet: <https://www.mongodb.com/blog/post/sega-hardlight-migrates-to-mongodb-atlas-simplify-ops-improve-experience-mobile-gamers>

Edgegap. **ONLINE GAMING CONNECTIVITY REPORT 2022**. 2022. Acesso em: 18/10/2022. Disponível na Internet: <https://edgegap.com/lag_report-2022>.

Epic Games. **Fortnite**. 2017. Acesso em: 25/08/2022. Disponível na Internet: <https://www.epicgames.com/fortnite/en-US/home>.

Epic Games. **Postmortem of Service Outage 4/11/2018 - 4/12/2018**. 2018. Acesso em: 25/08/2022. Disponível na Internet: <https://www.epicgames.com/fortnite/pt-BR/news/postmortem-of-service-outage-4-12>.

Epic Games. **Fortnite now has over 350 million registered players!** 2020. Acesso em: 25/08/2022. Disponível na Internet: <https://twitter.com/fortnitegame/status/1258079550321446912>.

EVSTIGNEEV, P. **Postbird**. 2014. Acesso em: 02/09/2022. Disponível na Internet: <https://github.com/paxa/postbird>.

GEE, W.; KRISHNAMURTHY, S. Global operations: How zynga scales its databases to support millions of players (presented by amazon). Game Developers Conference. 2019. Disponível na Internet: <https://www.gdcvault.com/play/1026141/Global-Operations-How-Zynga-Scales>.

GeeksForGeeks. **What makes Python a slow language?** 2021. Acesso em: 03/10/2022. Disponível na Internet: <https://www.geeksforgeeks.org/what-makes-python-a-slow-language/>.

GOOGLE. **Firebase Realtime Database**. 2012. Acesso em: 25/08/2022. Disponível na Internet: <https://firebase.google.com/products/realtime-database>.

GOOGLE. **Firebase Remote Config**. 2016. Acesso em: 25/08/2022. Disponível na Internet: <https://firebase.google.com/docs/remote-config>.

GOOGLE. **Halfbrick increases revenue by 16% with Remote Config personalization**. 2021. Acesso em: 25/08/2022. Disponível na Internet: <https://firebase.google.com/use-cases/halfbrick-personalization/?hl=en-us>.

GRAY, J. The transaction concept: Virtues and limitations. **Proceedings of the 7th International Conference on Very Large Databases**, VLDB Endowment, v. 24, n. 10, p. 144–154, 1981.

Halfbrick Studios. **Fruit Ninja**. 2010. Acesso em: 25/08/2022. Disponível na Internet: <https://play.google.com/store/apps/details?id=com.halfbrick.fruitninjafree&hl=pt_BR&gl=US>.

KURABAYASHI, S. Create a 20 times faster database engine optimized to mmogs. Game Developers Conference. 2016. Disponível na Internet: <https://www.gdcvault.com/play/1023800/Create-a-20-Times-Faster>.

LEAVITT, N. Will nosql databases live up to their promise? **Computer**, v. 43, n. 2, p. 12–14, 2010.

MongoDB Inc. **MongoDB**. 2009. Acesso em: 25/08/2022. Disponível na Internet: <https://www.mongodb.com>.

MongoDB Inc. **Read Concern**. 2022. Acesso em: 12/09/2022. Disponível na Internet: <https://www.mongodb.com/docs/manual/reference/read-concern/>.

MongoDB Inc. **Replication**. 2022. Acesso em: 12/09/2022. Disponível na Internet: <https://www.mongodb.com/docs/manual/replication/#redundancy-and-data-availability>.

RONACHER, A. **Flask**. 2010. Acesso em: 02/09/2022. Disponível na Internet: <https://flask.palletsprojects.com/en/2.2.x/>.

SAFKO, L.; BRAKE, D. K. **The Social Media Bible: Tactics, Tools, and Strategies for Business Success**. [S.l.: s.n.], 2009.

SALEN, K.; ZIMMERMAN, E. **Rules of Play: Game Design Fundamentals**. [S.l.]: MIT Press, 2003. ISBN 978-0-262-24045-1.

SEGA Corporation. **Sonic Forces**. 2018. Acesso em: 25/08/2022. Disponível na Internet: <https://play.google.com/store/apps/details?id=com.sega.sprint&hl=pt_BR&gl=US>.

Statista Research Department,. **Number of online console gamers in the United States from 2014 to 2020**. 2016. Acesso em: 17/10/2022. Disponível na Internet: <https://www.statista.com/statistics/521822/number-of-online-console-gamers-in-the-us/>.

STONEBRAKER, M. **PostgreSQL**. 1996. Disponível na Internet: <https://www.postgresql.org/>.

TAKAHASHI, D. **Zynga picks Unity Technologies to provide ads across its games**. 2017. Acesso em: 25/08/2022. Disponível na Internet: <https://venturebeat.com/2017/08/02/zynga-picks-unity-technologies-to-provide-ads-across-its-games/>.

TAYLOR, R. W.; FRANK, R. L. Codasyl data-base management systems. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 8, n. 1, p. 67–103, march 1976. ISSN 0360-0300. Disponível na Internet: <https://doi.org/10.1145/356662.356667>.

The Pokémon Company. **O Site Oficial de Pokémon**. 1998. Acesso em: 08/09/2022. Disponível na Internet: <https://www.pokemon.com/br/>.

The Pokémon Company. **Business Summary**. 2021. Acesso em: 08/09/2022. Disponível na Internet: <https://corporate.pokemon.co.jp/en/aboutus/figures/>.

The PostgreSQL Global Development Group. **What Is PostgreSQL?** 2009. Acesso em: 25/08/2022. Disponível na Internet: <https://www.postgresql.org/docs/current/intro-whatis.html>.

TSICHRITZIS, D. C.; LOCHOVSKY, F. H. Hierarchical data-base management: A survey. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 8, n. 1, p. 105–123, march 1976. ISSN 0360-0300. Disponível na Internet: <https://doi.org/10.1145/356662.356667>.

WEILBACHER, M. Dedicated servers in gears of war 3: Scaling to millions of players. Game Developers Conference. 2012. Disponível na Internet: <https://www.gdcvault.com/play/1015337/Dedicated-Servers-In-Gears-of>.

Zynga Inc. **Zynga Poker**. 2007. Acesso em: 25/08/2022. Disponível na Internet: <https://zyngapoker.com/pt-br/index.html>.

Zynga Inc. **Senior Database Administrator**. 2022. Acesso em: 25/08/2022. Disponível na Internet: <https://www.zynga.com/job-listing/senior-database-administrator-p210714/>.