

TCC UFFS

Fair: Uma Ferramenta Genérica para Migração de Dados entre SGBDs

Fair: A Generic Tool for Data Migration between SGBDs

Alexx Antoniollo  [Universidade Federal da Fronteira Sul | antoniolloalex@gmail.com]

 UFFS, Universidade Federal da Fronteira Sul, Rodovia SC 484 - Km 02, Chapecó, SC, 89815-899, Brasil.

Resumo. A crescente adoção de diferentes modelos de bancos de dados tem tornado a migração de dados uma atividade fundamental para garantir interoperabilidade, escalabilidade e adequação às necessidades específicas das aplicações modernas. Entretanto, as diferenças estruturais entre bancos relacionais e não relacionais tornam esse processo complexo, exigindo mecanismos capazes de preservar a integridade dos dados e adaptar as estruturas entre os sistemas envolvidos. Neste trabalho é apresentada a Fair, uma ferramenta de migração de dados projetada para operar de forma genérica e independente da tecnologia utilizada. A arquitetura proposta utiliza uma camada de adaptação baseada no padrão Adapter e um modelo intermediário responsável por desacoplar os sistemas de origem e destino, permitindo maior extensibilidade e reutilização do processo de migração. Os experimentos realizados com as bases de referência DVDrental e Chinook 1.4 demonstraram a capacidade da ferramenta em realizar migrações de PostgreSQL para MongoDB e Neo4j com preservação das estruturas e relacionamentos dos dados originais. Os resultados indicaram estabilidade temporal na abordagem voltada ao MongoDB, com tempos de migração próximos de 5 a 6 segundos mesmo com o aumento do volume de dados, enquanto a abordagem para Neo4j apresentou crescimento progressivo do tempo de processamento.

Abstract. The increasing adoption of different database models has made data migration a fundamental activity to ensure interoperability, scalability, and suitability to the specific needs of modern applications. However, the structural differences between relational and non-relational databases make this process complex, requiring mechanisms capable of preserving data integrity and adapting structures between the involved systems. This work presents Fair, a data migration tool designed to operate in a generic and technology-independent manner. The proposed architecture employs an adaptation layer based on the Adapter design pattern and an intermediate model responsible for decoupling source and target systems, allowing greater extensibility and reusability of the migration process. Experiments conducted using the benchmark datasets DVDrental and Chinook 1.4 demonstrated the tool's ability to perform migrations from PostgreSQL to MongoDB and Neo4j while preserving the structures and relationships of the original data. The results indicated temporal stability in the MongoDB-oriented approach, with migration times remaining close to 5 to 6 seconds even as data volume increased, whereas the Neo4j approach exhibited a progressive growth in processing time.

*dvdrental*¹ e o *Chinook 1.4*²

Palavras-chave: Migração de Dados, PostgreSQL, MongoDB, Neo4j, Bancos de Dados NoSQL, Modelo Intermediário, ETL

Keywords: Data Migration, PostgreSQL, MongoDB, Neo4j, NoSQL Databases, Intermediate Model, ETL

Recebido/Received: DD Month YYYY • Aceito/Accepted: DD Month YYYY • Publicado/Published: DD Month YYYY

1 Introdução

A crescente variedade de SGBDs e a expansão do uso de modelos NoSQL têm tornado cada vez mais frequente a necessidade de migração de dados entre diferentes tecnologias. Enquanto bancos de dados relacionais, como o PostgreSQL, continuam amplamente utilizados devido à sua maturidade e consistência, soluções orientadas a documentos e grafos, como MongoDB e Neo4j, oferecem características importantes de flexibilidade estrutural, escalabilidade e melhor adequação a determinados tipos de aplicações [Sadallage and Fowler, 2012]. Entretanto, as diferenças existentes entre esses modelos de dados tornam o processo de migração uma tarefa complexa, exigindo mecanismos capazes de preservar a integridade das informações e adaptar adequadamente as estruturas entre os sistemas envolvidos.

Diversos trabalhos têm sido propostos para tratar a migração entre bancos de dados relacionais e NoSQL ([Hanine *et al.*, 2015], [Mahmood, 2018], [Karnitis and Arnicans,

2015] [Ramzan *et al.*, 2019], [Bansel *et al.*, 2016]). Por exemplo, Hanine *et al.* [2015] apresentam uma abordagem baseada no modelo ETL para conversão entre MySQL e MongoDB. Já Mahmood [2018] propõe um algoritmo híbrido para migração de SQL Server para MongoDB, combinando estratégias *embedded* e *linked*. Karnitis and Arnicans [2015] utilizam uma metodologia baseada em múltiplos níveis de abstração e documentos XML intermediários, enquanto Ramzan *et al.* [2019] empregam uma arquitetura baseada em metamodelos e mecanismos de validação para migração entre SQL Server e Oracle NoSQL. Apesar dos avanços obtidos, grande parte dessas abordagens é voltada para cenários específicos e apresenta limitações quanto à extensibilidade e ao suporte a diferentes tecnologias de BD.

Diante desse contexto, este trabalho apresenta a ferramenta Fair, uma solução de migração de dados desenvolvida para operar de maneira genérica e independente do SGBD utilizado. A arquitetura proposta é baseada no padrão Adap-

ter [Bansel *et al.*, 2016] e utiliza um modelo intermediário capaz de desacoplar os bancos de origem e destino, permitindo a implementação de algoritmos de transformação especializados para diferentes modelos de dados. Como prova de conceito, foram desenvolvidos os algoritmos *Rel2Doc* [Garbin *et al.*, 2024], voltado para migrações de PostgreSQL para MongoDB, e *Graphened* [Bortoluzzi *et al.*, 2026], destinado à conversão de bases relacionais para grafos no Neo4j.

Para avaliar a proposta, foram realizados experimentos utilizando as bases de dados *dvdrental* e *Chinook*. Os resultados demonstraram que a ferramenta foi capaz de realizar corretamente a transformação e a migração dos dados para diferentes SGBDs. No caso do MongoDB, as tabelas relacionais foram consolidadas em um número reduzido de coleções, com tempos de execução da ordem de segundos. Já na migração para o Neo4j, foram gerados nós e relacionamentos preservando a semântica estrutural do esquema original. Esses resultados evidenciam a viabilidade da arquitetura proposta e a flexibilidade proporcionada pela utilização de um modelo intermediário associado a algoritmos especializados de transformação.

2 Fundamentação Teórica

Nesta seção, serão abordados noções necessárias para uma melhor compreensão deste trabalho. A seguir, apresentam-se conceitos relacionados à BD NoSQL e BDR, migração de dados, normalização e desnormalização.

2.1 Bancos de Dados Relacionais

Antes do surgimento dos SGBDs, as organizações utilizavam simples arquivos de sistema operacional para armazenar e manipular informações [Silberschatz, 2021]. Cada aplicação possuía seus próprios arquivos e programas específicos de acordo com as necessidades e práticas adotadas pela organização. Na medida que novas necessidades surgiam, era preciso fazer a criação novos programas e arquivos, mas, devido à arquitetura baseada em arquivos independentes, a complexidade dos sistemas aumentavam e diversas desvantagens surgem, como: redundância e inconsistência de dados, dificuldade de acesso e isolamento das informações, problemas de integridade e atomicidade, anomalias no acesso concorrente e problemas na segurança dos dados.

Os BDRs (Bancos de Dados Relacionais) são sistemas baseados no modelo relacional, no qual os dados são organizados em tabelas que modelam entidades e suas relações [Sadalage and Fowler, 2012]. Cada tabela é composta por colunas e linhas. Esse modelo utiliza linguagens específicas para manipulação e criação de dados, conhecidas respectivamente como DML (*Data Manipulation Language*) e DDL (*Data Definition Language*), sendo SQL a linguagem mais amplamente utilizada nos bancos de dados relacionais. Sua ampla adoção deve-se à padronização promovida por organismos internacionais, à sua capacidade de expressar operações complexas de consulta e manipulação de dados de forma declarativa e ao suporte oferecido pelos principais SGBDs relacionais [Silberschatz, 2021].

Apesar de seus benefícios, um projeto de esquema mal estruturado também apresenta impactos no desempenho do BD Sadalage and Fowler [2012]. Em modelos excessivamente normalizados, a fragmentação dos dados em várias ta-

id	nome	telefone	endereco
1	Fábio	(21)XXXX-XXXX	Rua 6, Copacabana, Rio de Janeiro/RJ - 22080-040
2	Zezinho	[(61)XXXX-XXXX (61)XXXX-XXXX]	Quadra 102, Conjunto 8, Samambaia Sul, Brasília/DF - 72300-015
3	Maria	(11)XXXX-XXXX	Rua 47, Jardim São Paulo, São Paulo/SP - 08465-312

Figura 1. Tabela não de acordo com a FN1 Ferreira [2023]

id	nome	logradouro	bairro	cidade	estado	cep
1	Fábio	Rua 6	Copacabana	Rio de Janeiro	RJ	22080-040
2	Zezinho	Quadra 102 Conjunto 8	Samambaia Sul	Brasília	DF	72300-015
3	Maria	Rua 47	Jardim São Paulo	São Paulo	SP	08465-312

id_usuario	telefone
2	(61)XXXX-XXXX
2	(61)XXXX-XXXX
1	(21)XXXX-XXXX
3	(11)XXXX-XXXX

Figura 2. Tabelas normalizada após a aplicação FN1 Ferreira [2023]

belas aumenta a necessidade de operações de junção, o que degrada o tempo de resposta em consultas complexas ou em ambientes com grande volume de leitura. Por outro lado, a desnormalização busca reduzir esse custo ao concentrar informações em menos tabelas, o que acelera consultas, mas introduz redundâncias que tornam as operações de escrita mais pesadas e elevam o risco de inconsistências [Silberschatz, 2021].

2.1.1 Normalização e Desnormalização

A normalização é um processo fundamental no projeto de BDRs, cujo principal objetivo é eliminar redundâncias, evitar anomalias de inserção, exclusão e atualização, e garantir a consistência dos dados. Esse processo consiste em decompor relações complexas em estruturas menores, preservando a integridade e os relacionamentos entre os dados. A normalização é guiada por um conjunto de regras chamadas formas normais, que determinam o grau de organização de um esquema de BD, serão apresentadas as três principais formas normais Silberschatz [2021].

A primeira forma normal (1FN) estabelece que todos os domínios de atributos devem ser atômicos, ou seja, cada campo deve conter apenas um valor indivisível. Isso significa que não se devem armazenar listas, conjuntos ou estruturas compostas dentro de um único atributo.

As Figuras 1 e 2 ilustram o processo de normalização para a 1FN. Na tabela superior, observa-se uma estrutura inicial em que cada usuário possui seus dados pessoais e endereço, porém as informações de telefone podem ocorrer mais de uma vez para um mesmo usuário, conforme pode ser visto na tupla com ID 2. Para atender à 1FN. As tabelas apresentadas na Figura 2 mostram o esquema após a aplicação da 1FN.

A segunda forma normal (2FN) e a terceira (3FN) são correlatas. A 2FN trata das dependências parciais, onde nenhum atributo não-chave pode depender de apenas uma parte da chave primária, ou seja, todos os atributos devem depender da chave primária inteira. Já a 3FN é relacionado a dependências transitivas, ela estipula que nenhum atributo não-chave pode depender de outro atributo não-chave, ou

id	id_servico	servico	total_horas	valor_hora	subtotal
1	0001	Designer de Site	12	70,00	840,00
2	0002	Desenvolvimento de Software	50	90,00	4.500,00
3	0001	Consultoria	5	100,00	500,00

Figura 3. Tabela não normalizada Ferreira [2023]

id	id_servico	total_horas	subtotal
1	0003	5	500,00
2	0001	50	840,00
3	0002	12	4.500,00

id_servico	servico	valor_hora
0001	Designer de Site	70,00
0002	Desenvolvimento de Software	90,00
0003	Consultoria	100,00

Figura 4. Tabelas normalizadas Ferreira [2023]

seja, todos os atributos devem depender diretamente da chave primária. Um esquema está na 2FN e na BCNF quando todos os seus atributos não-chave dependem inteiramente da chave primária. A Figura 3 ilustra a aplicação da 2FN e BCNF, na qual deve ser realizada a eliminação das dependências parciais por meio da decomposição de tabelas, perceba que o campos *servico* e *valor_hora* não dependem unicamente do atributo *id*, e sim são dependentes do *id_servico*.

Desta forma, na tabela original, que possui os atributos *id_servico*, *servico* e *valor_hora* estavam repetidos para cada registro da tabela principal, gerando redundância e riscos de inconsistência. Após a normalização, a estrutura foi reorganizada em duas tabelas distintas como apresentado na Figura 4.

A desnormalização tem seu objetivo o oposto da normalização. Uma vez que, ao normalizar o banco busca reduzir redundâncias, com a desnormalização, é introduzido mais redundâncias com a intenção de melhorar o desempenho do sistema Hollingsworth [2017]. Em bancos de dados NoSQL, especialmente nos modelos orientados a documentos e às famílias de colunas, a desnormalização é uma prática comum, uma vez que esses sistemas priorizam a performance de leitura e a escalabilidade horizontal. Essa abordagem permite consultas mais rápidas e simplifica o acesso a dados frequentemente relacionados, ainda que aumente a necessidade de cuidados adicionais para manter a consistência Sadalage and Fowler [2012].

Na Figura 5 é apresentado uma tabela de acompanhamento de horas de funcionários em projetos. Inicialmente a tabela encontra-se desnormalizada e em seguida é feito o processo de normalização Gomes [2024].

Na Figura 5, a normalização foi aplicada para corrigir uma tabela que misturava informações de funcionário, projeto e do trabalho realizado no projeto. Para resolver isso, a



Figura 5. Normalização Gomes [2024]

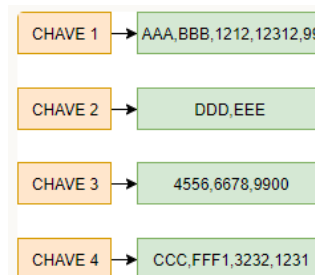


Figura 6. Chave-valor Almeida [2024]

tabela foi decomposta em três novas: uma contendo apenas os dados do funcionário, outra contendo apenas os dados do projeto e uma terceira representando a relação trabalho entre ambos.

Na aplicação das FNs nesse processo, a 1FN garantiu que todos os dados fossem atômicos e organizados sem repetições de grupos de atributos. A 2FN foi atendida ao separar atributos que dependiam apenas de parte da chave composta, criando tabelas próprias para cada entidade e evitando dependências parciais. A BCNF removeu dependências transitivas ao assegurar que cada atributo estivesse diretamente ligado à chave primária de sua tabela.

2.2 Bancos de Dados NoSQL

O termo NoSQL surge do fato de não limitar os bancos apenas ao SQL, o que sugere um afastamento dos SGBDs relacionais tradicionais baseadas em SQL, o que abandona os modelos de esquemas convencionais Sadalage and Fowler [2012]. Os bancos NoSQL ganharam ênfase no mercado devido às necessidades da *big data*, o que motivou a procura por novas formas para o armazenamento de dados, capazes de aumentar a produtividade ao facilitar o acesso às informações, mesmo que exija escalar versões do BD em mais de uma mesma máquina por meio de clusterização.

Cada banco NoSQL possui um modelo de dados diferente, são agrupados em quatro categorias: chave-valor, documento, família de colunas e grafo.

No modelo chave-valor é feito uma associação entre uma chave única e seu valor no BD criando assim uma grande tabela hash, uma estrutura de dados que organiza chaves de buscas a valores Florencio *et al.* [2017].

Na Figura ?? nota-se que cada uma das chaves esta associada a um valor específico, e que com essa chave é possível consultar seu valor armazenado. Além disso, é importante ressaltar, que geralmente, não é possível realizar buscas diretamente nos valores associados as chaves.

Em uma base de dados orientada a grafos, os dados são representados por meio de nós, enquanto as arestas indicam as relações existentes entre esses nós. As propriedades são atributos específicos tanto dos nós como das arestas que fornece informações adicionais para compreensão das relações presentesBatra and Tyagi [2012]. Na Figura ?? nota-se um grafo que ilustra diferentes tipos de relacionamentos entre entidades, representadas por nós conectados por arestas rotuladas. Cada nó corresponde a uma pessoa, objeto ou conceito, enquanto as arestas indicam o tipo de relação existente entre eles. No centro do grafo, observa-se Bob, que mantém múltiplas conexões, funcionando como o principal ponto de interação entre as entidades. Bases de dados orientadas a grafo são eficientes quando se lida com áreas onde a interconec-

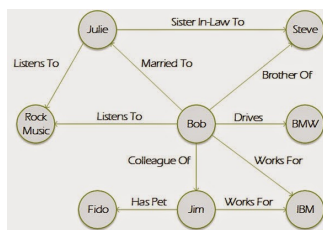


Figura 7. Grafo Hogg [2013]

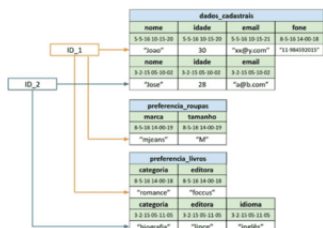


Figura 8. Família de Colunas Lima [2021]

tividade dos dados tem um papel central, já que a estrutura dos dados facilita consultas complexas sobre relacionamento e controle ligação.

Em um BD orientado a família de colunas a estrutura de valores é definida como um conjunto de colunas predefinidas, cujos dados são armazenados em conjunto. A inclusão de colunas é um processo eficiente realizado linha por linha, onde as linhas possuem um conjunto diferente de colunas, ou até mesmo nenhuma. Essa flexibilidade permite que as tabelas permaneçam vazias sem gerar custos de armazenamento para valores nulos Redmond *et al.* [2018].

A Figura 8 apresenta um exemplo de como os dados são organizados em um BD orientado a colunas. Nesse modelo, cada usuário é identificado por uma chave, e a partir dessa chave são agrupados diferentes conjuntos de informações, chamados de famílias de colunas.

Documentos dentro de um BD orientado a documentos são semelhantes às linhas em bancos de dados relacionais, mas são mais flexíveis, uma vez que não possuem esquemas, elas tem seu salvamento em estruturas flexíveis, que são obtidas por meio de dados em formatos semi estruturados como JSON, BSON e XML Nayak *et al.* [2013].

Conforme ilustrado na Figura 9, nesse modelo as informações são organizadas em documentos independentes, geralmente estruturados em JSON ou BSON, que podem ser relacionados entre si por meio de identificadores.

No exemplo apresentado, há um *user document* que representa um usuário e contém atributos básicos, como o identificador e o nome de usuário. Outros documentos, como *contact document* e *access document*, armazenam informações complementares.

2.3 Migração de Dados

Na migração de dados, as informações vindas do sistema legado precisam ser transferidas para o novo sistema. Apesar

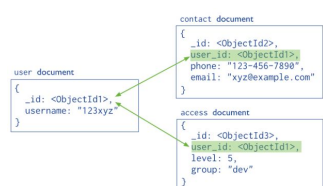


Figura 9. Orientado a Documentos

da importância desse processo, o conhecimento sobre o processo de migração é limitado Tanya and Azizah [2023].

A migração é o processo que visa migrar dados formatados de uma estrutura de origem para uma estrutura de destino, sendo que ambas as estruturas diferem em nível conceitual e/ou físico Matthes *et al.* [2011].

Diversas abordagens e métodos para fazer a migração entre SGBDs são documentados na literatura acadêmica, tais como: conversão de esquemas, meta-modelagem e ETL (Extract, Transform, Load).

No processo de conversão de esquemas proposto por [Fong *et al.*, 2006], são gerados documentos XML sem redundância e com a preservação de todas as restrições do banco de dados, permitindo a conversão dos dados para o formato XML. Após essa etapa, ocorre a transformação dos dados utilizando o modelo gerado a partir do esquema Entidade-Relacionamento Estendido (EER) adotado pelos autores.

A meta-modelagem consiste na criação manual do mapeamento de conversão do BD existente para um esquema XML previamente definido do que vai ser migrado. A conversão de dados de RDB para NoSQL ocorre em duas etapas: Na primeira parte o objetivo é em obter a estrutura física dos dados do BDR e elevar o nível semântico dessa estrutura, o que cria o meta-modelo. Nessa fase, é criada o escopo e as regras da migração. Na segunda etapa, realiza-se a conversão dos dados, em que as informações do banco relacional são transformadas para o modelo NoSQL, de acordo com as estruturas e diretrizes estabelecidas no metamodelo e no escopo definidos anteriormente.

A técnica da migração baseada no modelo ETL, começa com o processo de extração (Extract) das estruturas e informações de um BDR, identificando tabelas, atributos, chaves primárias e relacionamentos. Em seguida, acontece a etapa de transformação (Transform), esses elementos são convertidos em estruturas compatíveis com o modelo do BD de destino. Por fim, na fase de carga (Load), os dados transformados são inseridos no novo sistema, criando automaticamente o novo esquema de armazenamento.

De forma geral, as principais estratégias de migração discutidas incluem a conversão de esquemas proposta por [Fong *et al.*, 2006], que transforma o modelo relacional em documentos XML preservando as restrições e a estrutura do banco de dados; a meta-modelagem, que estabelece manualmente um mapeamento conceitual entre o banco existente e o modelo de destino; e a abordagem ETL, que organiza o processo em etapas de extração, transformação e carga dos dados. Considerando a clareza metodológica e a adequação ao contexto deste trabalho, será utilizada uma abordagem baseada no modelo ETL para conduzir o processo de migração.

3 Trabalhos Relacionados

Nesta seção, o estudo apresentará e discutirá trabalhos relevantes ao tema que demonstram similaridade com o projeto em desenvolvimento, especialmente no que diz respeito às metodologias e técnicas utilizadas para a ferramenta proposta. Para a identificação desses trabalhos, foram realizadas buscas no Google Scholar e Semantic Scholar utilizando a consulta (NoSQL data migration) or (database migration). A

partir dessa pesquisa, foram encontrados 16 artigos, dentre os quais destacam-se os seguintes:

O trabalho de [Hanine *et al.*, 2015], expõe uma ferramenta para migração de uma base de dados entre MySQL e MongoDB. O processo da ferramenta proposta utiliza o modelo ETL (Extract, Transform, Load), que consiste em extrair as estruturas e dados do sistema relacional, transformar em um modelo orientado a documentos e, por fim, fazer a inserção dos dados no ambiente NoSQL.

A ferramenta, desenvolvido em Java, automatiza a conversão de tabelas e relacionamentos em coleções e *arrays* de documentos embutidos. Essa abordagem elimina a necessidade de joins e cria uma coleção desnormalizada o que adapta os dados a um formato mais comumente visto em modelos NoSQL.

O trabalho de [Mahmood, 2018] apresenta uma ferramenta em VB.NET para migração de um banco de dados SQL Server para MongoDB. A metodologia proposta consiste em iterar pelas tuplas de cada tabela do SQL Server, mapeando os dados das tuplas para documentos que serão armazenados em coleções no MongoDB. O algoritmo oferece três modos de estruturação: *Embedded*, onde documentos filhos são embutidos aos documentos pais; *Linked*, em que cada tabela se torna uma nova coleção e documentos filhos referenciam documentos pais por chaves estrangeiras; e *Auto*, que mescla as duas abordagens utilizando *Embedded* se o tamanho máximo do documento é tolerável com base em uma configuração e *Linked* em casos em que o valor seja ultrapassado.

O trabalho de [Karnitis and Arnicans, 2015], propõe uma metodologia semi automática, o que permite ajustes manuais durante a processo. O estudo utiliza SQLServer como BD de origem Clusterpoint como destino. A abordagem utilizada é baseada em modelagem multinível, que organiza os dados em vários níveis de abstração antes da migração.

No primeiro nível lógico, os dados do modelo físico são reinterpretados, identificando objetos simples, complexos e relações *N:N*, e permite adição de atributos virtuais.

No segundo nível lógico, cada *Table-like Structures* (TLS) central serve de foco para gerar árvores hierárquicas de estruturas relacionadas. São definidos as estruturas internas dos documentos, estruturando os dados de forma agregada considerando os relacionamentos e aninhamento.

O terceiro nível lógico, corresponde à fase de transformação, são criados templates de documentos XML, que servem como etapa intermediária do protótipo final e validação antes da conversão para o formato do banco NoSQL.

No quarto nível lógico, ocorre a migração dos dados para o BD NoSQL, os templates XML são convertidos para o formato nativo do BD destino.

O trabalho de [Ramzan *et al.*, 2019] propõe uma metodologia semi automática para migração de bancos de dados relacionais, especificamente SQLServer, para o Oracle, utiliza uma abordagem baseada em *Model Driven Architecture* (MDA). A modelagem é realizada por meio de metamodelos: o metamodelo relacional do SQLServer já padronizado e um metamodelo inicial do Oracle criado pelos autores para possibilitar a transformação. A migração utiliza o motor de

transformação SiTra³, que transforma o modelo de entrada no modelo de saída. Cada tabela, linha e coluna do SQL-Server é convertida em tabelas, pares chave-valor e campos no Oracle, e substituindo operações de join por relações pai-filho.

O segundo módulo da abordagem trata da integridade dos dados migrados, utilizando técnicas de detecção e correção de erros. O sistema identifica inconsistências, dados incompletos, duplicados e registros irrelevantes por meio de parsing, clustering baseado em distância euclidiana e checagem de checksums, permite correções automáticas ou manuais.

O trabalho de [Bansel *et al.*, 2016] propõe um *framework* para migração de diferentes bancos de dados NoSQL em nuvem, com foco em MongoDB, Azure Table e Neo4j. Os autores apresentam uma abordagem baseada em metamodelos, que permite padronizar e classificar os dados antes da migração. O framework possibilita duas modalidades para migração: direta MongoDB convertido para Neo4j e intermediária, na qual os dados do banco de origem são traduzidos para um modelo Azure Table intermediário antes de serem migrados para o banco de final. Esse modelo intermediário permite uma maior flexibilização e robustez na organização das entidades e propriedades, permitindo extensibilidade para suportar outros tipos de bancos, porém implica em redução de desempenho.

Para facilitar a compreensão das características e diferenças entre os principais modelos de migração de dados analisados, a Tabela 1 apresenta uma comparação clara entre os artigos estudados. Diversas técnicas e metodologias foram propostas para a transformação entre diferentes SGBDs, cada uma abordando de forma distinta o processo de migração.

No que diz respeito à modelagem de bancos de dados orientados a documentos, observam-se duas abordagens principais: *Embedded* e *Linked*. A abordagem *Linked* cria uma *collection* equivalente para cada tabela existente no banco de origem, enquanto a *Embedded* busca reduzir a normalização das tabelas, aninhando as tabelas referenciadas dentro da tabela principal (modelo de agregados).

No entanto, o uso excessivo de *embedding* pode resultar em *collections* muito extensas, o que pode ultrapassar os limites de armazenamento e comprometer o desempenho do BD. Para contornar essa limitação, [Mahmood, 2018] propôs uma abordagem híbrida, que permite a utilização de três modos de operação: *Linked*, *Embedded* ou Automático. Neste último, o sistema adota *Embedded* por padrão, mas, ao detectar que o tamanho excede um limite predefinido, alterna automaticamente para o modo *Linked*, equilibrando eficiência e escalabilidade.

A migração de dados pode ser realizada de duas formas principais: direta e indireta. Na migração direta, os dados são transferidos diretamente do SGBD de origem para o SGBD de destino, sem a utilização de estruturas intermediárias. Esse modelo é mais simples e rápido, porém apresenta menor flexibilidade e maior risco de incompatibilidades, especialmente quando os sistemas envolvidos possuem modelos de dados distintos. Já a migração indireta introduz um modelo intermediário de padronização, que atua como uma

³<https://github.com/bbordbar/sitra>

Tabela 1. Comparação entre os modelos de migração de dados analisados

Aspecto	Hanine (2015)	Mahmood (2018)	Karnitis (2015)	Ramzan (2019)	Bansel (2016)
Modelagem	Embedded	Híbrida	Embedded	Embedded	Grafo
Migração	Direta	Direta	Indireta	Indireta	Híbrida
Validação	Não	Não	Parcial	Completo	Não
Origem	MySQL	SQLServer	SQLServer	SQLServer	MongoDB
Destino	MongoDB	MongoDB	Clusterpoint	Oracle	Neo4j
Mod. Inter.	Não	Não	Sim	Sim	Sim
Índices	Não	Não	Não	Sim	Não

camada padronizada entre os dois sistemas.

Nessa abordagem, os dados são primeiramente convertidos para um formato comum, facilitando sua adaptação a diferentes SGBDs e permitindo maior controle sobre a consistência, transformação e validação dos dados. [Karnitis and Arnicans, 2015], fizeram uma validação de forma simples e estrutural, sendo realizada no modelo intermediário em XML gerado a partir do banco relacional. Nesse caso, é validado se não existe nenhum atributo inconsistente, se todos os relacionamentos para as chaves estão corretamente definidos, e que não tenha perda de dados ou ambiguidade. Já no trabalho de [Ramzan *et al.*, 2019], a validação torna-se mais complexa, incorporando um módulo de validação de distancia euclidiana com clusterings, regras de consistência multidimensional, parsing e checagem de checksums. Essas validações verificam a estrutura a qualidade a proximidade dos dados, similitude de registros e limites de consistência.

Os artigos selecionados para compor a tabela têm em comum o foco na migração de bancos de dados relacionais para modelos NoSQL, especialmente os orientados a documentos ou baseados em chave-valor. A única exceção entre os trabalhos analisados é o trabalho de [Bansel *et al.*, 2016] que propõe a migração do BD MongoDB para o Neo4j, deslocando o foco da conversão relacional-NoSQL para uma transição entre dois tipos distintos de bancos NoSQL.

De modo geral, a maioria dos artigos analisados não considera a migração de índices como um fator relevante; entretanto, o trabalho de Ramzan *et al.* [2019] destaca a importância desse critério. Assim, o sistema proposto neste estudo procura combinar os aspectos positivos das abordagens existentes, propondo uma metodologia adaptativa compatível com diferentes SGBDs, incorporando a consideração de índices e um módulo de validação de dados.

4 Fair

A ferramenta *Fair* consiste em um sistema de migração de dados entre Sistemas de Gerenciamento de Bancos de Dados (SGBDs), projetado para operar de forma genérica e independente do tipo de banco utilizado. Sua arquitetura é baseada em um motor de migração desacoplado dos BDs por meio de adaptadores específicos para cada SGBD. O processo de migração inicia com a extração dos dados e de seus metadados a partir do banco de origem, seguida da conversão dessas informações para um modelo intermediário padronizado, responsável por representar entidades, atributos e relacionamentos de forma independente do modelo de dados. Em seguida, são aplicadas regras de mapeamento e transformação específicas para o banco de destino, para gerar estruturas adequadas e a posterior importação dos dados.

4.1 Visão Geral da Arquitetura

A Figura 10 apresenta uma visão geral da arquitetura do sistema de migração. Ela é composta pela interface de acesso, responsável pela interação com o usuário; pelo módulo de conexão com os SGBDs, que estabelece a comunicação com as bases de origem e destino; pelo mecanismo de exportação dos dados e pelo modelo intermediário, utilizado como representação comum entre os diferentes modelos de dados. A arquitetura também inclui o componente de mapeamento, encarregado de definir as correspondências entre os elementos dos SGBDs, o algoritmo de transformação, responsável pela adaptação dos dados ao modelo de destino, e o módulo de importação, que realiza a inserção das informações no SGBD de destino. A descrição detalhada do funcionamento desses componentes é apresentada a seguir por meio do fluxo de execução do sistema.

A arquitetura do sistema segue um fluxo sequencial. Inicialmente, o usuário configura os SGBDs de origem e destino, estabelece as conexões e seleciona as estruturas e mapeamentos a serem migrados. O processo é composto pelas etapas de configuração, transformação e inserção dos dados no SGBD de destino. Cada uma dessas etapas é detalhada a seguir.

A etapa de conexão é responsável por estabelecer a comunicação com os SGBDs de origem e destino por meio dos adaptadores do sistema. Nessa fase, o usuário fornece os parâmetros de acesso ao BD, como *host*, porta, usuário, senha e nome da base, além de opções específicas suportadas pelo SGBD. Em seguida, o sistema executa uma consulta simples para validar as informações fornecidas e verificar se a conexão foi estabelecida corretamente.

A etapa de exportação é responsável pela extração da base de dados do SGBD de origem, são exportados os dados armazenados e o *schema* da aplicação. Nessa fase, são coletadas as informações estruturais necessárias, como tabelas, colunas, chaves primárias, relacionamentos e demais metadados relevantes, além dos próprios registros da base. Assim, o conteúdo é organizado de forma compatível com o modelo intermediário utilizado pelo sistema.

A etapa de mapeamento é responsável por apresentar ao usuário todas as estruturas exportadas da base de dados, com o intuito de realizar ajustes antes da migração definitiva. Nessa fase, tabelas e atributos são listados para que o usuário possa definir quais elementos deverão ou não ser migrados. Além disso, o sistema possibilita a renomeação de tabelas e colunas conforme a necessidade da migração.

O modelo intermediário é responsável por transformar os dados extraídos já mapeados uma representação abstrata independente do tipo de SGBD utilizado. Após o mapeamento as correções especificadas pelo usuário, dados e *sche-*

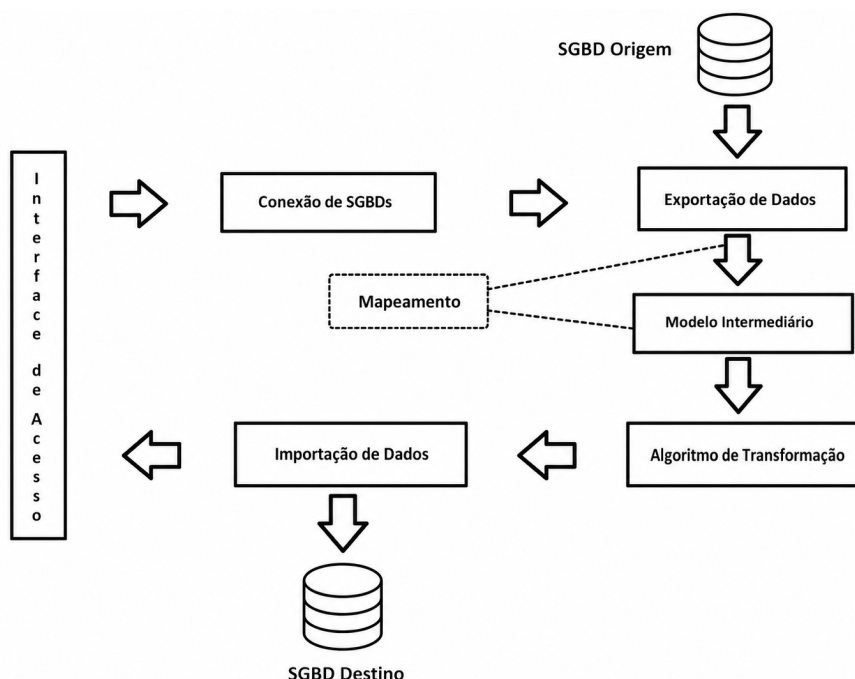


Figura 10. Arquitetura Fair

mas são organizados no modelo intermediário adotado pelo sistema, para que nas próximas etapas possa ser manipuladas as informações de maneira padronizada e compatível.

A etapa de transformação é responsável pela aplicação das regras de conversão, normalização e adaptação dos dados contidos no modelo intermediário. Nessa fase, o sistema apresenta ao usuário uma lista com os algoritmos de transformação disponíveis, contendo o nome e a descrição de cada um deles, permitindo a seleção mais adequada para o seu caso específico. Após a escolha, o algoritmo selecionado é aplicado sobre o modelo intermediário, o que realiza as alterações necessárias para converter a estrutura e os dados em um novo modelo compatível com as características do BD de destino.

A etapa de importação é responsável pela inserção das informações processadas no SGBD de destino. Nessa fase, o modelo intermediário transformado é utilizado como base para recriar a estrutura da base de dados no sistema de destino. Após a criação da estrutura, os dados transformados são inseridos no banco de dados.

Após a conexão, os dados são extraídos do SGBD de origem mediante de um processo de exportação. Esses dados não são transferidos diretamente para o destino, mas sim convertidos para um modelo intermediário, que atua como uma representação unificada dos dados e de seus respectivos esquemas.

O modelo intermediário é composto por duas estruturas principais: uma estrutura de dados, responsável por armazenar os registros de cada tabela, e uma estrutura de esquema, que descreve os metadados associados a essas tabelas. Para cada entidade são armazenadas informações como chave primária, colunas e chaves estrangeiras, além da tabela e o atributo referenciado.

Além dessas estruturas, o modelo intermediário possui um conjunto opcional de informações de mapeamento, uti-

```

100, 2 meses ago | 1 autor (100)
export interface DatabaseAdapter {
  connect(): Promise<void>;

  read(collection: string, params?: string): Promise<any[]>;
  write(collection: string, data: any[]): Promise<void>;

  listTables(schemaName: string): Promise<string[]>;
  createTable(tableName: string): Promise<void>;
  listColumns(table: string, schemaName?: string): Promise<string[]>;
  transformData(
    data: ModeloIntermediario,
    algorithm: number,
  ): ModeloIntermediario;
  listAlgorithms(): AlgorithmInfo[];
  listSchema?(schemaName: string): Promise<Record<string, TableSchema>>;
}

```

Figura 11. Interface DatabaseAdapter

lizado principalmente para algoritmos de migração para BD orientados a grafos. Essas informações permitem identificar se determinada estrutura deve ser representada como nó ou aresta.

Em seguida, a ferramenta permite manipular os nomes e seleção dos dados do SGBD no processo de mapeamento, responsável por ajustar manualmente as estruturas do banco de origem com o modelo intermediário. Esse mapeamento permite lidar com diferenças como nomes de atributos e seleção de tabelas.

Uma vez estruturados no modelo intermediário, são listados os algoritmos de transformação implementados no adapter do SGBD de destino, assim, é feito a transformação do modelo intermediário para realizar a inserção.

4.2 Camada de Adaptação

A comunicação com diferentes SGBDs é realizada por meio de uma camada de adaptação baseada no padrão de projeto *Adapter*. Essa camada define uma interface genérica denominada *DatabaseAdapter* 11, responsável por padronizar as operações básicas de interação com os bancos de dados, independentemente da tecnologia utilizada. Entre suas principais responsabilidades estão o estabelecimento da conexão

```

async importData(
  adapter: DatabaseAdapter,
  data: ModeloIntermediario,
): Promise<void> {
  if (adapter instanceof Neo4JAdapter) {
    await adapter.write("", [data]);
  } else {
    for (const [table, records] of Object.entries(data.data)) {
      await adapter.write(table, records);
    }
  }
}

```

Figura 12. Implementação da função importData

com o SGBD, a leitura e escrita de dados, além da obtenção de informações estruturais, como tabelas, colunas e esquemas do banco.

Além das operações de acesso aos dados, a interface também define métodos relacionados ao processo de transformação do modelo intermediário, permitindo que cada adaptador implemente algoritmos específicos de conversão e manipulação de dados conforme as características do banco utilizado.

4.3 Motor de Migração

O componente responsável por orquestrar o processo de migração é o `MigrationEngine`. Ele coordena as etapas de criação do modelo intermediário e a importação dos dados no destino, além de permitir o download do modelo intermediário para exportação.

As principais funcionalidades do MigrationEngine concentram-se em duas etapas fundamentais do processo de migração. A primeira delas é a função `createModel`, responsável pela geração do modelo intermediário a partir dos dados extraídos do banco de origem. Essa função realiza a leitura das tabelas e colunas selecionadas pelo usuário, aplica os mapeamentos de tabelas e colunas definidos durante a configuração da migração e constrói uma representação padronizada dos dados, independente do SGBD utilizado.

A segunda funcionalidade é a importação dos dados para o banco de destino. A Figura 12 apresenta um exemplo da implementação desse algoritmo no motor de migração. Nessa etapa, o `MigrationEngine` utiliza o adaptador correspondente ao SGBD selecionado para percorrer as estruturas presentes no modelo intermediário e delegar as operações de escrita de acordo com as características do sistema de destino.

A função `importData` é responsável por realizar a inserção dos dados no banco de destino. Para isso, ela recebe o adaptador correspondente ao SGBD utilizado e percorre os registros presentes no modelo intermediário, delegando ao adaptador a escrita das informações nas estruturas de armazenamento do banco de destino. Dessa forma, a lógica de inserção permanece desacoplada do motor de migração, o que permite que diferentes bancos de dados sejam suportados por meio de implementações específicas dos adaptadores. No caso do `Neo4JAdapter`, os dados são convertidos diretamente em grafos, enquanto, nos bancos relacionais, os registros são processados relação por relação.

5 Experimentos

Os experimentos realizados neste trabalho tiveram como objetivo avaliar o processo de migração de dados entre diferentes modelos de bancos de dados. Para isso, foram consideradas as bases de dados de referência *DVD Rental* e

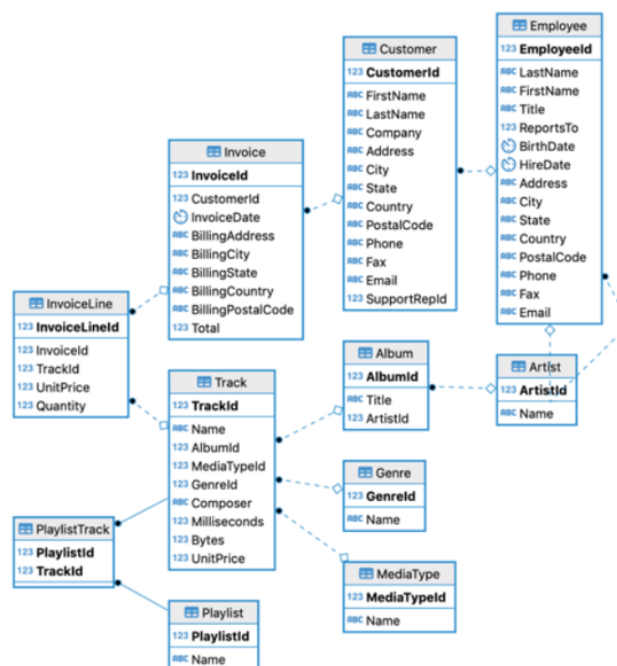


Figura 13. Modelo Entidade Relacionamento Neon [2024]

Chinook 1.4, ambas originalmente armazenadas em PostgreSQL. As migrações foram conduzidas em dois cenários distintos: PostgreSQL para MongoDB, que representa uma migração para um banco orientado a documentos, e PostgreSQL para Neo4j, que representa uma migração para um banco orientado a grafos. A avaliação foi dividida em diferentes tamanhos de conjuntos de dados, para analisar o comportamento da solução em cenários com volumes crescentes de informação. Inicialmente, os experimentos foram executados com um quarto da base de dados original. Em seguida, foram realizados testes com metade da base, posteriormente com a base completa e, por fim, com versões expandidas por meio da inserção de registros adicionais.

O ambiente de experimentação utilizado neste trabalho consiste em um notebook equipado com um processador AMD Ryzen 9 7940HS, que possui GPU integrada Radeon 780M Graphics. O sistema operacional utilizado é o Windows 11. O equipamento conta com 16 GB de memória RAM DDR4 e armazenamento SSD de 512 GB. Para o desenvolvimento dos experimentos, optou-se pela utilização da linguagem TypeScript, em conjunto com o framework Next.js para a construção da aplicação.

5.1 Validação de conceito

A validação de conceito tem como objetivo verificar se a camada de migração do Fair é capaz de transferir corretamente os dados entre os SGBDs, de acordo com as etapas definidas na arquitetura — seleção dos bancos de dados de origem e destino, extração, modelo intermediário, mapeamento, transformação e importação.

O processo inicia com a extração dos dados a partir do PostgreSQL, com o conjunto de dados DVD Rental definido como base de origem, por meio de consultas para leitura das tabelas e seus respectivos metadados - chaves primárias, colunas e relacionamentos implícitos. Esses dados foram então convertidos para o modelo intermediário definido pela ferramenta.



Figura 14. Fair Seleção de Tabelas

A Figura 13 apresenta o modelo Entidade-Relacionamento do banco de dados DVD Rental, com as principais entidades, seus atributos e os relacionamentos entre elas. O diagrama oferece uma visão da estrutura lógica do banco de dados e evidencia as conexões estabelecidas por meio de chaves primárias e estrangeiras.

Na sequência, foi realizada a etapa de mapeamento, na qual as estruturas extraídas foram apresentadas ao sistema para definição de ajustes manuais, como seleção de tabelas, exclusão de atributos e renomeação de colunas. Após essa etapa, os dados passaram por normalização e organização no modelo intermediário, garantindo consistência estrutural para as próximas fases. Vale destacar que, devido à limitação do Neo4j em armazenar diretamente arquivos binários de imagem, a coluna *picture* presente no BDR foi excluída durante o mapeamento para evitar problemas de compatibilidade.

A etapa de transformação foi responsável pela aplicação dos algoritmos de conversão de modelo, sendo composta por duas abordagens distintas, conforme o tipo de destino da migração: o algoritmo *Rel2Doc*, voltado para a conversão de PostgreSQL para MongoDB, e o algoritmo *Graphened*, utilizado na conversão de PostgreSQL para Neo4j.

O algoritmo *Rel2Doc* Garbin *et al.* [2024] tem como objetivo estruturar os dados relacionais em formato de documentos. Inicialmente, ele identifica tabelas de junção com base em características estruturais, como a presença de duas chaves estrangeiras e baixa quantidade de atributos. Essas tabelas são utilizadas para reconstruir relações N:N no formato de documentos aninhados. Em seguida, o algoritmo percorre cada tabela principal e realiza a agregação de dados relacionados, tratando diferentes tipos de relacionamento: relações 1:N, em que uma entidade pai agrega múltiplos registros filhos; relações 1:1, em que uma entidade incorpora diretamente outra única entidade; e relações N:N, que são reconstruídas a partir das tabelas de junção. As tabelas de referência também são incorporadas como subdocumentos.

O algoritmo *Graphened* Bortoluzzi *et al.* [2026] realiza a transformação do modelo relacional em um grafo otimizado para Neo4j, seguindo três etapas principais: identificação de tabelas de junção, unificação de entidades e definição de mapeamento com criação de arestas.

Na primeira etapa, o algoritmo percorre o esquema relacional e identifica tabelas de junção com base em critérios

estruturais, como a presença de duas chaves estrangeiras, ausência de referências externas e quantidade limitada de atributos não primários. Essas tabelas são classificadas como candidatas a se tornarem arestas no grafo - relações N:N entre entidades.

A segunda etapa consiste na unificação de tabelas, na qual tabelas fortemente relacionadas podem ser absorvidas por um nó principal, desde que respeitados limites configuráveis de atributos máximos *MAX_UATT* e evitando a criação de ciclos na estrutura do grafo. Para cada relação de chave estrangeira 1:1 ou 1:N, o algoritmo avalia se é possível consolidar os registros, assim, os dados são agregados da tabela dependente ao nó pai.

Na terceira etapa, o algoritmo define o mapeamento final para o Neo4j. Cada tabela é classificada como nó ou aresta. Tabelas unificadas são registradas dentro do nó pai, e as chaves estrangeiras restantes são convertidas em arestas direcionadas. Caso a unificação tenha absorvido uma tabela de destino, a chave estrangeira é incorporada como atributo interno e não gera uma aresta separada.

Por fim, na etapa de importação, os dados transformados foram inseridos nos SGBDs de destino. No caso do MongoDB, os documentos foram armazenados em coleções correspondentes às tabelas de origem. No caso do Neo4j, a operação é um pouco mais complexa devido à necessidade de preservar a estrutura de grafo e otimizar a escrita em banco.

O processo para o Neo4j inicia com a criação de um mapeamento entre as tabelas originais e os rótulos de nós, considerando tabelas que foram unificadas durante a etapa de transformação pelo algoritmo *Graphened*. Em seguida, os nós são inseridos em lotes de tamanho 1000, com a cláusula *MERGE* para evitar duplicação e *SET* para atualizar atributos existentes.

Após a inserção dos nós, os relacionamentos são criados a partir das chaves estrangeiras mapeadas como arestas. Cada aresta é representada por um par de nós, e seu tipo é definido a partir do campo correspondente à chave estrangeira. Assim como na criação dos nós, os relacionamentos também são inseridos em lotes. Ao término da migração, foi realizada uma etapa de validação por meio da comparação entre a base relacional de origem e a base em Neo4j, verificando-se a quantidade de nós, relacionamentos e referências associadas às chaves estrangeiras. Essa análise permitiu confirmar que as estruturas e relações presentes no PostgreSQL foram preservadas no banco de destino.

5.2 Avaliação de Desempenho

Para avaliar o desempenho das abordagens de migração implementadas, foram utilizados dois conjuntos de dados públicos: o *dvdrental*⁴ e o *Chinook 1.4*⁵. Os experimentos foram conduzidos com quatro variações de tamanho para cada base de dados: um quarto da base original, metade, a base completa e uma versão expandida, obtida por meio da inserção de novos registros.

⁴Disponível em: <https://neon.com/postgresql/getting-started/sample-database>

⁵Disponível em: <https://github.com/lerocha/chinook-database/releases>

```

_id: ObjectId('6a22b455d4f70e8c89035e1d')
film_id: 133
title: "Chamber Italian"
description: "A Fateful Reflection of a Moose And a Husband who must Overcome a Monk..."
release_year: 2006
language_id: 1
rental_duration: 7
rental_rate: "4.99"
length: 117
replacement_cost: "14.99"
rating: "NC-17"
last_update: "2013-05-26T17:50:58.951Z"
special_features: Array (1)
fulltext: "'chamber':1 'fate':4 'husband':11 'italian':2 'monkey':16 'moos':8 'mu..."
actor: Array (7)
category: Array (1)
language: Object

```

Figura 15. Collection film

5.2.1 dvdrental

O conjunto de dados *dvdrental* é composto originalmente por 15 tabelas relacionais, que modelam um sistema de locação de filmes com entidades como filmes, atores, clientes e alugueis. Essa estrutura relacional serviu como entrada para ambas as abordagens de migração avaliadas.

Na abordagem *Rel2Doc*, o algoritmo realizou o agrupamento das tabelas relacionais e a inserção dos dados no MongoDB. As 15 tabelas originais foram consolidadas em apenas 6 *collections*, independentemente do volume de dados processado. A Figura 15 apresenta a estrutura da collection *films*. Em termos de volume, a versão com um quarto da base originou 10.080 documentos a partir de 12.624 registros no PostgreSQL; a versão com metade da base resultou em 19.389 documentos a partir de 23.541 registros. A base completa gerou 37.424 documentos a partir de 44.820 registros. E a base expandida produziu 102.424 documentos a partir de 129.821 registros.

Na abordagem *Graphened*, a migração para o Neo4j preservou as entidades e seus relacionamentos na forma de nós e arestas. A versão com um quarto da base resultou em 11.010 nós e 25.673 relacionamentos. A versão com metade da base gerou 20.321 nós e 50.756 relacionamentos. A base completa produziu 38.356 nós e 99.305 relacionamentos. E a base expandida resultou em 118.356 nós e 234.305 relacionamentos.

Em relação ao desempenho temporal, a abordagem *Rel2Doc* manteve um tempo de migração aproximado de 6 segundos em todos os cenários, demonstrando estabilidade frente ao crescimento do volume de dados. Já a abordagem *Graphened* apresentou crescimento mais acentuado: a versão com um quarto da base foi migrada em cerca de 25 segundos; a versão com metade da base levou aproximadamente 90 segundos; a base completa demandou cerca de 6 minutos; e a base expandida exigiu aproximadamente 27 minutos e 10 segundos. A Figura 16 apresenta os resultados obtidos no experimento.

5.2.2 Chinook 1.4

O conjunto de dados *Chinook 1.4* é composto originalmente por 11 tabelas relacionais, com um sistema de loja de música digital com entidades como artistas, álbuns, faixas, clientes e faturas. Essa estrutura serviu como entrada para as mesmas duas abordagens avaliadas no conjunto anterior.

Na abordagem *Rel2Doc*, as 11 tabelas relacionais foram consolidadas em 5 *collections* no MongoDB. A versão com um quarto da base originou 1.198 documentos. A versão com metade da base resultou em 2.660 documentos. A base completa gerou 6.510 documentos. E a base expandida produziu

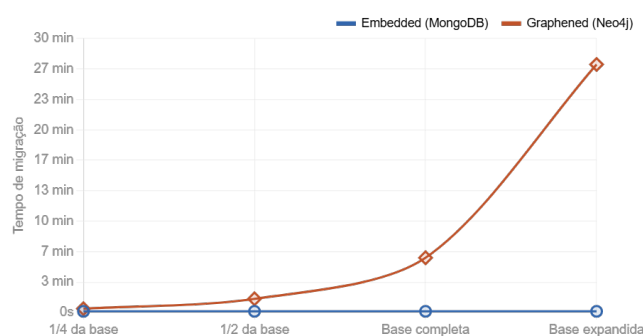


Figura 16. DvdRental Dataset

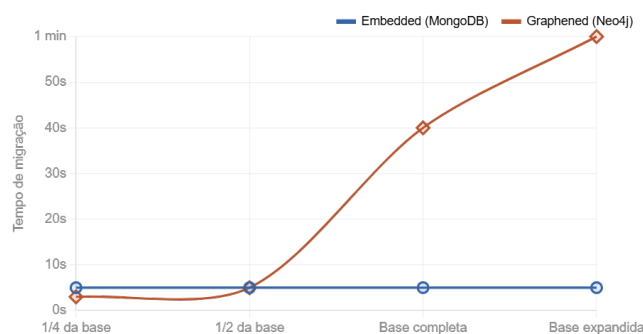


Figura 17. Chinook Dataset

13.012 documentos.

Na abordagem *Graphened*, a migração para o Neo4j gerou estruturas de grafo proporcionais ao volume de dados. A versão com um quarto da base resultou em 1.165 nós e 2.818 relacionamentos. A versão com metade da base produziu 2.260 nós e 5.640 relacionamentos. A base completa gerou 4.652 nós e 11.327 relacionamentos. E a base expandida resultou em 9.307 nós e 22.713 relacionamentos.

Em relação ao desempenho temporal, a abordagem *Rel2Doc* manteve um tempo de migração aproximado de 5 segundos em todos os cenários, reforçando o padrão de estabilidade observado no conjunto anterior. A abordagem *Graphened* também apresentou crescimento progressivo: a versão com um quarto da base foi migrada em cerca de 3 segundos; a versão com metade da base levou aproximadamente 5 segundos; a base completa demandou cerca de 40 segundos; e a base expandida exigiu aproximadamente 1 minuto. A Figura 17 apresenta os resultados obtidos no experimento.

6 Conclusão

Este trabalho apresentou a implementação da ferramenta Fair, uma arquitetura para migração de dados entre diferentes SGBDs. A solução foi desenvolvida com base em um motor de migração desacoplado dos SGBDs por meio do padrão *Adapter*, o que possibilitou a extração e a importação dos dados de forma independente das tecnologias envolvidas. Além disso, a utilização de um modelo intermediário padronizado permitiu representar entidades, atributos e relacionamentos em uma estrutura comum antes da aplicação das transformações necessárias para cada banco de destino.

Os experimentos realizados demonstraram a capacidade da ferramenta Fair de migrar dados entre diferentes SGBDs, com preservação das informações e adaptação das estruturas aos modelos de destino. A utilização de algoritmos especí-

ficos para cada tipo de banco evidenciou a versatilidade da abordagem proposta.

Os resultados obtidos com as bases *dvdrental* e *Chinook* mostraram que o algoritmo *Embedded* apresentou baixos tempos de execução e reduziu a quantidade de estruturas necessárias no MongoDB por meio da agregação de tabelas relacionais em coleções. Por sua vez, o algoritmo *Graphened* permitiu representar explicitamente as relações entre os dados por meio da criação de nós e arestas no Neo4j.

Os experimentos também confirmaram a viabilidade do uso de um modelo intermediário como mecanismo de desacoplamento entre os bancos de origem e destino.

6.1 Trabalhos Futuros

Como trabalhos futuros, pretende-se ampliar a ferramenta por meio da inclusão de suporte a um maior número de SGBDs, bem como a novos elementos e estruturas específicas de cada modelo de dados. Essa expansão permitirá aumentar a abrangência da arquitetura e sua aplicabilidade em mais cenários de migração.

Outra possibilidade consiste na implementação de mecanismos de migração paralela, para que as etapas de extração, transformação e importação sejam executadas de forma concorrente, com intuito de melhorar o desempenho.

Referências

- Almeida, V. (2024). Bancos de dados nosql orientados a pares de chave-valor: Um olhar sob a ótica dos concursos. <https://blog.grancursosonline.com.br/bancos-de-dados-nosql-orientados-a-pares-de-chave-valor/> [Acessado em: (06/27/2026)].
- Bansel, A., González-Vélez, H., and Chis, A. E. (2016). Cloud-based nosql data migration. In *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, pages 224–231. IEEE.
- Batra, S. and Tyagi, C. (2012). Comparative analysis of relational and graph databases. *International Journal of Soft Computing and Engineering (IJSCE)*, 2(2):509–512.
- Bortoluzzi, I., Duarte, D., Schreiner, G. A., and Feitosa, S. (2026). Graphened: A tool for converting relational databases into graph databases. Artigo submetido para publicação no JIDM.
- Ferreira, F. J. L. (2023). Banco de dados - modelagem de dados.
- Florencio, D. A., de Oliveira, D. R. F., Freitas, E. L. S. X., and de Souza, F. d. F. (2017). Which fits better? a comparative analysis about nosql key-value databases. *IEEE Latin America Transactions*, 15(11):2251–2256.
- Fong, J., Fong, A., Wong, H., and Yu, P. (2006). Translating relational schema with constraints into xml schema. *International Journal of Software Engineering and Knowledge Engineering*, 16(02):201–243.
- Garbin, T. S., Duarte, D., and Schreiner, G. A. (2024). Rel2doc: Migrating data from relational to document-oriented databases. In *Simpósio Brasileiro de Banco de Dados (SBBDD)*, pages 107–112. SBC.
- Gomes, P. C. T. (2024). Normalização de dados: Como aplicar em um banco de dados? <https://www.datageeks.com.br/normalizacao-de-dados/> [Acessado em: (11/17/2025)].
- Hanine, M., Bendarag, A., and Boutkhoul, O. (2015). Data migration methodology from relational to nosql databases. *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, 9(12):2566–2570.
- Hogg, A. (2013). Whiteboard it – the power of graph databases. <https://www.computerweekly.com/feature/Whiteboard-it-the-power-of-graph-databases> [Acessado em: (06/27/2026)].
- Hollingsworth, M. (2017). Data normalization, denormalization, and the forces of darkness. *vol. V. 0.4, ed.*
- Karnitis, G. and Arnicans, G. (2015). Migration of relational database to document-oriented database: Structure denormalization and data transformation. In *2015 7th international conference on computational intelligence, communication systems and networks*, pages 113–118. IEEE.
- Lima, L. O. (2021). O que é nosql? <https://blog.grancursosonline.com.br/o-que-e-nosql/> [Acessado em: (06/27/2026)].
- Mahmood, A. A. (2018). Automated algorithm for data migration from relational to nosql databases. *ALNAHRAIN JOURNAL FOR ENGINEERING SCIENCES*, 21(1):60–65.
- Matthes, F., Schulz, C., and Haller, K. (2011). Testing & quality assurance in data migration projects. In *2011 27th IEEE international conference on software maintenance (ICSM)*, pages 438–447. IEEE.
- Nayak, A., Poriya, A., and Poojary, D. (2013). Type of nosql databases and its comparison with relational databases. *International Journal of Applied Information Systems*, 5(4):16–19.
- Neon (2024). postgresql dvd rental er diagram - tutorial. <https://neon.com/postgresqtutorial/printable-postgresql-sample-database-diagram.pdf> [Acessado em: (06/27/2026)].
- Ramzan, S., Bajwa, I. S., Ramzan, B., and Anwar, W. (2019). Intelligent data engineering for migration to nosql based secure environments. *IEEE Access*, 7:69042–69057.
- Redmond, E., Perkins, L., and Wilson, J. (2018). Seven databases in seven weeks: a guide to modern databases and the nosql movement.
- Sadalage, P. J. and Fowler, M. (2012). Nosql distilled. <https://martinfowler.com/books/nosql.html?> [Acessado em: (11/17/2025)].
- Silberschatz, Korth, S. (2021). *Sistema de Banco de Dados 5ª edição*. Elsevier / Campus.
- Tandya, W. and Azizah, F. N. (2023). Migration of relational database to nosql document-oriented database. In *2023 IEEE International Conference on Data and Software Engineering (ICoDSE)*, pages 180–185. IEEE.