Banco de Dados Relacional *versus* Banco de Dados Não-Relacional: abordagem aplicada em um estudo de caso

Dafne Dayane Medrado Santos Amaral[1], Michele Cristiani Barion[2]
Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas – Instituto Federal de Educação, Ciência e Tecnologia de São Paulo (IFSP) – *Campus* Hortolândia – São Paulo – SP – Brasil
dafnemedrado@gmail.com[1], michele_barion@hotmail.com[2]

Abstract. *With the evolution of technology and the increase of data generated by computerized devices, databases are essential for storing this growing volume of data. Today, the market has a variety of databases that perform this function, among them there are two models called relational and non-relational with different characteristics and priorities. Thus, this article reports through a case study implemented with the use of a relational SQL database, the migration process to a non-relational NoSQL model, bringing to achieve this goal, three proposals addressing in each of them advantages and disadvantages, as well as a future proposal to build a tool that enables the migration of a database. This is a work that has the database area as its main area of knowledge.*

Resumo. *Com a evolução da tecnologia e do aumento de dados gerados por dispositivos computadorizados, os bancos de dados são essenciais para o armazenamento desse crescente volume de dados. Hoje, o mercado possui uma variedade de bancos de dados que realizam essa função, dentre eles há dois modelos denominados relacional e não-relacional com características e prioridades distintas. Assim, o presente artigo relata através de um estudo de caso implementado com o uso de um banco de dados relacional SQL, o processo de migração para um modelo não-relacional NoSQL, trazendo para alcançar este objetivo, três propostas abordando em cada uma delas vantagens e desvantagens, além de uma proposta futura para construção de uma ferramenta que possibilite a migração de uma base de dados. Este é um trabalho que tem a área de banco de dados como principal área do conhecimento.*

1. Introdução

Os bancos de dados têm um papel muito importante em sistemas de informação, consistindo em uma coleção de dados associados a um mini-mundo[1] onde estes dados estão logicamente relacionados. A definição, o armazenamento e a manipulação dos dados são realizados por um sistema especializado denominado Sistema de Gerenciamento de Banco de Dados (SGBD).

Os SGBDs são baseados na teoria matemática das relações, na qual as bases de dados são chamadas de relacionais e a integridade se dá por meio de atributos, restrições de chaves e aplicação das regras de normalização. Conforme apresenta o *site* DB-Engines[2], há no mercado vários bancos de dados relacionais, podendo ser de uso gratuito ou pago. Em janeiro de 2021,

[1] Mini-mundo ou Universo de Discurso é a representação abstrata de uma pequena parte do mundo real, que é de grande interesse para uma aplicação. É o "problema" que devemos informatizar. Conceito extraído do *site* https://profsalu.files.wordpress.com/2014/05/modelagem-de-dados.pdf.

[2] O *site* https://db-engines.com/en/ranking mantém atualizada uma lista de *ranking*, considerando uma relação de bancos de dados usados mundialmente.

o DB-Engines apresentou os quatro primeiros SGBD's para bancos relacionais mais utilizados, sendo *Oracle, MySQL, Microsoft SQL Server* e *PostgreSQL*.

Desde então, as empresas adotam esse modelo de dados para os sistemas comerciais, porém com o surgimento da *Internet* e com a expansão de aplicações *web*, houve o aumento do fluxo de informações e transações desse tipo de banco de dados e, assim, trazendo preocupações aos profissionais que desenvolvem sistemas de informação, considerando a performance e a eficiência desses SGBD´s. Diante do exposto e, conforme apresenta o *site* da [DevMedia], em 2001 surge um conceito associado ao grande volume de dados denominado *big data*[3]. Este modelo traz uma proposta alternativa para se trabalhar com dados denominada não estruturado, com base no seu gerenciamento e armazenamento.

Assim, especialistas da comunidade de *software* livre e código aberto, buscaram o desenvolvimento de um banco de dados não-relacional surgindo assim, no final da década de 90, o *NoSQL*. Este novo modelo de dados teve como maior propósito resolver a questão da escalabilidade dos bancos relacionais, já que é uma ação que exige certa complexidade em solucioná-la e, consequentemente, tem alto custo para sua implementação e execução. Como apresenta o *site* DB-Engines, em janeiro de 2021 o *MongoDB*, que é um banco de dados não-relacional orientado a documentos, estava em 5º lugar como um dos *databases* mais usados nessa categoria.

Todavia, o *ranking* do DB-Engines ainda mostra um percentual alto de usuários que utilizam bancos de dados relacionais e nesta observação surge um questionamento: quais as possibilidades de migrar uma base relacional para não-relacional, seja através de uma aplicação disponibilizada para esse fim ou por meio de instruções executadas manualmente? Tendo em vista esta problemática, o presente trabalho objetiva usar um modelo relacional como um estudo de caso e apresentar diferentes maneiras de migrar um *database* no *MySQL* para um *database* não-relacional no *MongoDB*.

Para abordagem do objetivo exposto, além desta introdução, o presente trabalho está organizado em seis seções ordenadas em: fundamentação teórica, metodologia, desenvolvimento, conclusão, trabalhos futuros e relação das referências utilizadas tanto na pesquisa sobre as metodologias adotadas quanto das ferramentas que foram fundamentais para o desenvolvimento.

2. Referencial Teórico

Esta seção aborda a pesquisa bibliográfica realizada para gerar o embasamento teórico que fundamenta o uso de cada escolha para realização do presente trabalho.

2.1. Banco de Dados

[3] O conceito de *big data* não será explorado neste trabalho, porém o seu conceito está associado a um conjunto de ferramentas e práticas que gerenciam e analisam grandes volumes de dados, sendo que um dos motivos deste aumento significativo de armazenamento é o desenvolvimento da tecnologia, como é o caso do número de dados gerados diariamente por meios eletrônicos.

Segundo [Elmasri e Navathe 2018], um banco de dados é uma coleção de dados logicamente relacionados, sendo esses associados às características do mundo real denominados Mini-Mundo ou Universo de Discurso. Um banco de dados pode ser criado e mantido manualmente como um cartão de biblioteca ou computadorizado através dos Sistemas de Gerenciamento de Banco de Dados (SGBD).

O SGBD é um *software* que facilita o processo de definição e especificação dos tipos de dados, suas estruturas e restrições, denominados metadados; construção; armazenamento e manipulação; funções de consulta, e por fim o compartilhamento de banco de dados que permite o acesso simultâneo entre usuários e aplicações [HEUSER 2009]. Existem outras funções importantes associadas à administração de banco de dados, todavia elas não serão exploradas no contexto deste trabalho.

A Figura 1 apresenta as principais formas de acessar dados através de um SGBD, podendo ser através de instruções executadas diretamente por ele (considera-se usuários com conhecimentos técnicos para este fim) ou por meio de outros *softwares* que relacionem uma aplicação externa com uma base de dados (são mais utilizados por meio de usuários finais, a qual acessam informações através de opções programadas para este fim).

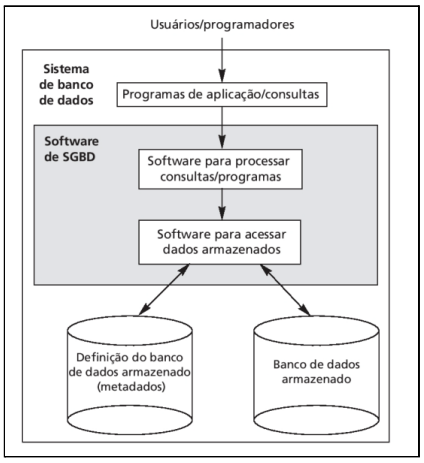


Figura 1. Ambiente de sistema de banco de dados [ELMASRI E NAVATHE 2018]

A arquitetura em um SGBD tem como principal objetivo separar as aplicações do usuário dos dados físicos. Como apresenta a Figura 2, essa arquitetura é subdividida em três níveis, sendo: o nível externo (1 - descreve os interesses de um grupo particular de usuários, sendo guiado pelos requisitos e usualmente representado em documentos textuais com a coleta através de questionários e/ou entrevistas com os chamados *stakeholders*[4]); o nível conceitual (2 - possui um esquema conceitual que descreve a estrutura do banco de dados, não priorizando detalhes de armazenamento físico e exibindo as descrições de entidades, relacionamentos, operações e restrições); e por fim o nível interno (3 - possui um esquema interno para descrever a estrutura de armazenamento físico do banco de dados por meio de um SGBD).

[4] Conforme apresenta [Sommerville 2019], os *stakeholders* em um projeto, são as pessoas ou organizações que são afetadas de forma positiva ou negativa pelos resultados do projeto.

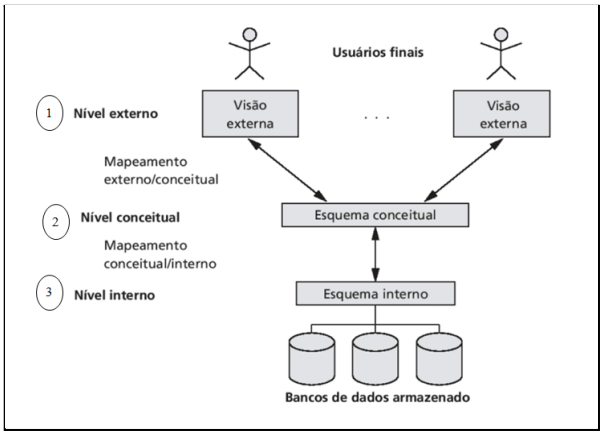


Figura 2. Arquitetura das Camadas de um SGBD [ELMASRI E NAVATHE 2018]

Enfatiza-se que há no mercado diferentes tipos de SGBD´s, sendo estes explorados nas subseções 2.1.1. e 2.1.2.

2.1.1. Modelo Relacional

O modelo de dados relacional, primeiramente foi proposto por Ted Codd da IBM no início da década de 70, sendo baseado no conceito matemático da teoria de relações. Todavia, esse modelo ficou conhecido pelas empresas em 1980, quando houveram as primeiras implementações comerciais como o sistema *SQL/*DS da IBM e o SGBD da Oracle [ELMASRI E NAVATHE 2018].

Considerando a popularidade dos diversos tipos de bancos de dados, o *site* [DBEngines 2021] apresenta mensalmente uma média que é calculada por várias variáveis que são coletadas por diferentes fontes e, assim, gerando um *ranking* dos sistemas mais populares nesta área[5]. Assim, conforme apresenta a Figura 3, no mês de março de 2021, os quatro primeiros colocados são do tipo relacional, sendo que o primeiro e segundo lugar apresentaram um aumento de popularidade em relação ao mês anterior.

Rank			DBMS	Database Model	Score		
Mar 2021	Feb 2021	Mar 2020			Mar 2021	Feb 2021	Mar 2020
1.	1.	1.	Oracle	Relational, Multi-model	1321.73	+5.06	-18.91
2.	2.	2.	MySQL	Relational, Multi-model	1254.83	+11.46	-4.90
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1015.30	-7.63	-82.55
4.	4.	4.	PostgreSQL	Relational, Multi-model	549.29	-1.67	+35.37

Figura 3. *Ranking* com foco nos bancos de dados relacionais [DBENGINES 2021]

Como apresentado na Figura 3, os bancos de dados relacionais mais usados na ordem crescente são: (1) Oracle, (2) *MySQL*, (3) Microsoft SQL Server, (4) *PostgreSQL*. Ressalta-se

[5] Para este levantamento, considera-se o número de menções em site de buscas como Google e Bing, a frequência de pesquisas e discussões técnicas de bancos nos sites StackOverflow e *DBA* Stack Exchange, a quantidade de vagas que menciona o sistema de banco de dados como Indeed e Simple Hided, profissionais na rede Linkedin que tem conhecimento nos sistemas e a relevância no Twitter sobre os sistemas. Com base nessas fontes é calculada uma média que mensura a popularidade de cada sistema, gerando-se, assim, o resultado do *ranking* que traz mensalmente os dez sistemas mais populares.

que o banco de dados *MySQL* e *PostgreSQL* são *open source*[6], sendo que Oracle e Microsoft são proprietários, ou seja, é requerido uma licença para utilização.

Os bancos de dados relacionais oferecem linguagens e interfaces para criação e manipulação de uma base de dados. A linguagem padrão utilizada nos bancos de dados desse modelo é a *Structured Query Language* ou mais conhecida como *SQL*.

No modelo relacional as tabelas são compostas por linhas que representam uma coleção de valores de dados. Para armazenar os dados neste modelo é necessário especificar as propriedades de cada campo que identifica um valor, além do cumprimento de restrições de integridade. É importante enfatizar que este modelo é chamado de relacional devido a relação que há entre os campos denominados chave primária (*primary key*) e chave estrangeira (*foreign key*) e, assim, cumprindo as regras de normalização para manter a integridade e evitar redundância de dados [ELMASRI E NAVATHE 2018].

A Figura 4 apresenta um exemplo de esquema relacional. Explorando o exemplo, [Heuser 2009] aborda que a "Atuação" é uma tabela que implementa o relacionamento que há entre as tabelas "Engenheiro" e "Projeto", sendo esta relação formada pelas chaves destas duas tabelas que associam "Atuação".

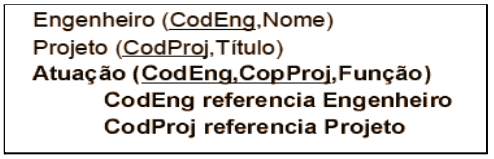


Figura 4. Exemplo esquema relacional [HEUSER 2009]

Do modelo relacional, surge o modelo físico que é a implementação do banco de dados em um SGBD no qual serão especificadas as estruturas e armazenamento dos dados. A implementação dessa etapa é realizada pela linguagem *Structure Query Language - SQL* (Linguagem de Consulta Estruturada). Esta é a linguagem padrão para implementação e manipulação de banco de dados relacionais.

2.1.1.1. Linguagem *SQL*

A linguagem *SQL* foi criada pela IBM Research em 1970, para um sistema de banco de dados experimental e agora é padrão para os SGBDs comerciais. A padronização dessa linguagem foi realizada pela *American National Standards Institute* (*ANSI*) e *Internacional Standards Organization* (*ISO*), utilizando a versão SQL:2016[7]. *SQL* não faz parte do conjunto das linguagens *case sensitive*.

A linguagem *SQL* é subdividida em algumas camadas, sendo: *Data Definition Language (DDL), Data Transaction Language (DTL), Data Controle Language (DCL)* e

[6] *Open source* é um termo que se refere ao *software open source* (*OSS*). Ele é um código projetado para ser acessado abertamente pelo público: todas as pessoas podem vê-lo, modificá-lo e distribuí-lo conforme suas necessidades. Conceito extraído do *site* https://www.redhat.com/pt-br/topics/open-source.

[7] Esta é a oitava revisão do padrão ISO e ANSI à linguagem *SQL*. O documento pode ser acessado através do *link* https://www.iso.org/obp/ui/#iso:std:63555:en.

Data Manipulation Language (DML). Como apresenta [Souza 2013], elas são representadas pelos seus conjuntos de comandos:

- *DDL*: é chamada de Linguagem de Definição de Dados. Esta camada associa comandos *SQL* para criação dos esquemas de banco de dados, sendo CREATE, DROP, TABLE.

- *DML*: é chamada de Linguagem de Manipulação de Dados. Após a criação dos esquemas de banco de dados, podem-se executar comandos para inserção, seleção, modificação ou remoção de dados. Para estas ações, na ordem, estão os comandos INSERT, SELECT, UPDATE e DELETE.

As camadas *DTL* e *DCL* não serão abordadas, pois não fazem parte do grupo de instruções que estão associadas a proposta deste trabalho.

Considerando a proposta do trabalho, a subseção 2.1.2.1. abordará as instruções fundamentais para gerar tabelas e seus relacionamentos, além da inserção de tuplas no modelo relacional.

2.1.1.1.1. Comandos para definição de esquemas de banco de dados

Conforme apresenta [Elmasri e Navathe 2018], na camada *DDL* estão as instruções utilizadas para criação de novas relações. Para isso, a linguagem *SQL* traz o comando CREATE, tendo como obrigatoriedade definir o nome da relação, especificar os atributos, os tipos, possíveis tamanhos e demais restrições iniciais. Os tipos de atributos podem variar de acordo com o tipo de SGBD, todavia os tipos mais conhecidos são INT, FLOAT, NUMERIC, DECIMAL, CHAR, VARCHAR e DATETIME (representando formatos para data e hora).

A Figura 5 apresenta um exemplo de tabela criada, considerando os atributos e suas propriedades, além da chave primária representada pelo atributo cpf.

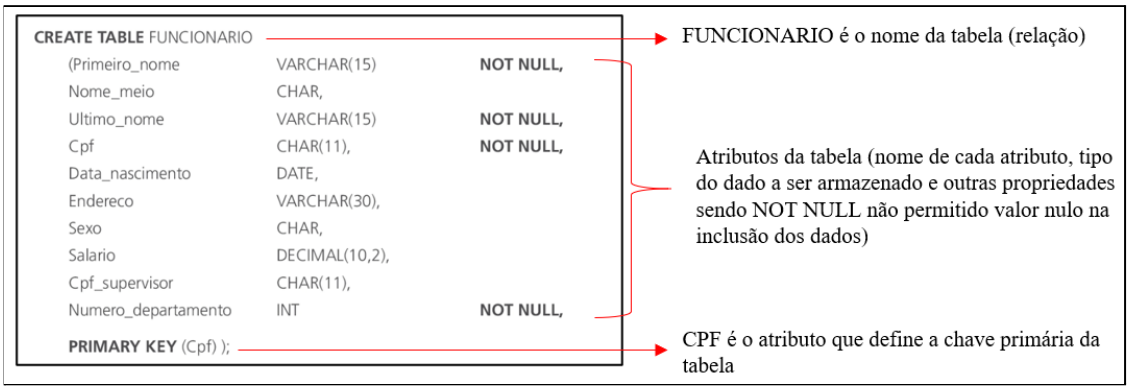


Figura 5. Exemplo de CREATE TABLE [ELMASRI E NAVATHE, 2018]

A Figura 6 também apresenta um modelo de tabela, porém além dos seus atributos, propriedades e chave primária, traz a chave estrangeira que faz a relação com a tabela funcionário (Figura 5), considerando a chave primária dessa tabela responsável por este relacionamento.

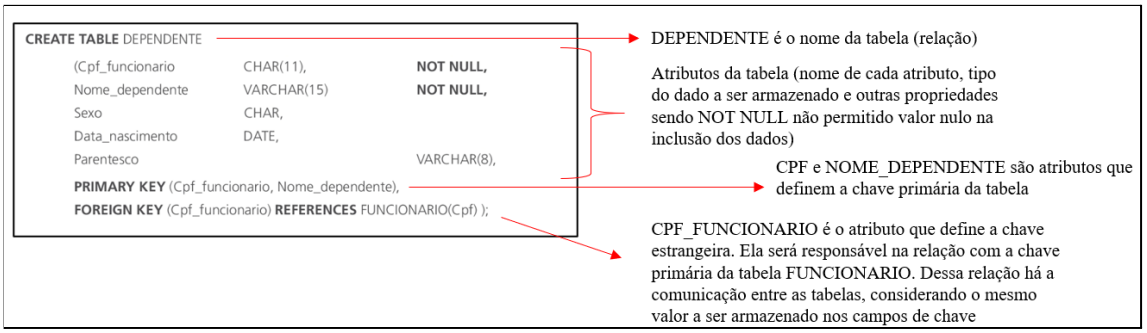


Figura 6. Exemplo de CREATE TABLE com chave estrangeira [ELMASRI E NAVATHE 2018]

2.1.1.1.2. Comando para inserção de dados em modelo relacional

Na camada *DML* o **INSERT** é a instrução utilizada para inserir os dados em uma tabela. A seguir um exemplo desta instrução (Figura 7), considerando a tabela apresentada na Figura 5.

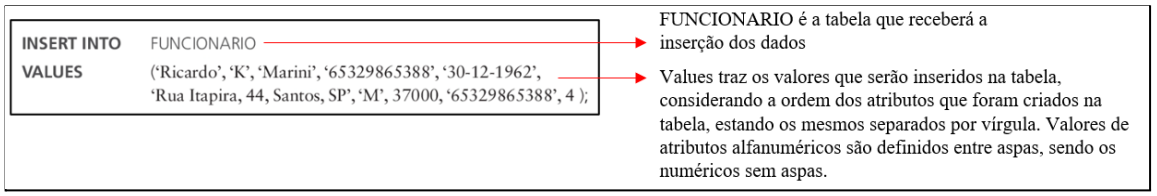


Figura 7. Exemplo de INSERT [ELMASRI E NAVATHE 2018]

2.1.1.1.3. Comando para consulta de dados em modelo relacional

O **SELECT** é uma instrução da camada *DML* que tem a função de consultar dados armazenados em uma ou várias tabelas. A Figura 8 traz um exemplo de consulta, considerando o exemplo da tabela criada através da Figura 7.

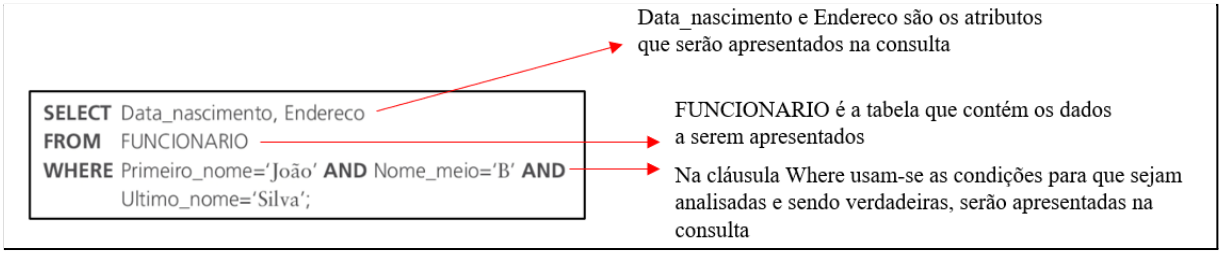


Figura 8. Exemplo de SELECT [ELMASRI E NAVATHE 2018]

Relata-se que, sendo um banco de dados relacional, as tabelas se comunicam através da relação que há entre as chaves associadas, ou seja, a chave primária de uma tabela com a chave estrangeira de outra tabela. Diante desta exploração e usando os modelos das tabelas apresentadas nas Figuras 5 e 6, pode-se solicitar uma listagem dos valores já inseridos com o nome do funcionário e seus respectivos dependentes, sendo assim, como seria na abordagem relacional, a execução dessa ação? A Figura 9 traz esta abordagem da relação entre tabelas através das chaves.

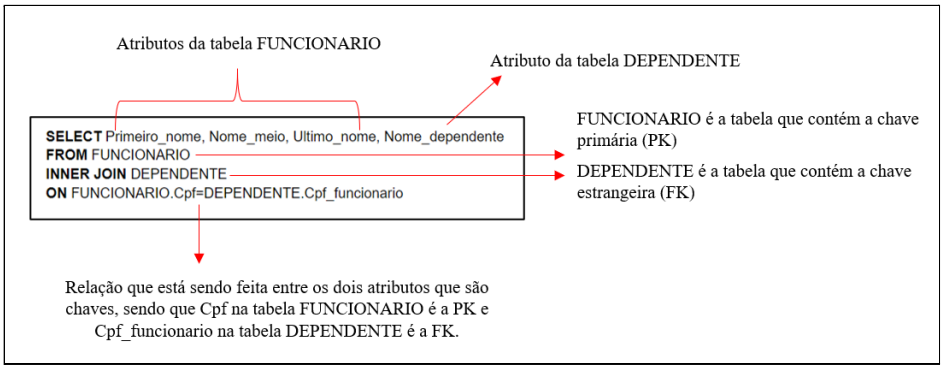


Figura 9. Exemplo de SELECT com relacionamento entre tabelas [ELMASRI E NAVATHE 2018]

Na Figura 9, observa-se que a relação entre as tabelas se faz através dos atributos Cpf da tabela FUNCIONARIO (sendo a chave primária) e Cpf_funcionario da tabela DEPENDENTE (sendo a chave estrangeira). Sendo o resultado dessa relação verdadeira, a instrução SELECT trará como resposta a relação do nome do funcionário e seu respectivo dependente. Esse conceito é a base dos modelos relacionais quanto a definição de tabelas e seus relacionamentos.

2.1.2. Modelo Não-Relacional

Segundo [Martin e Pramod 2013] o termo "*NoSQL*" surgiu no final década de 1990 com o nome de um banco de dados relacional *open source* por Carlo Strozzi, associando "*No* = Não" "*SQL* = Linguagem de Consulta Estruturada". É um modelo que tem um pouco mais de uma década, sendo que em 2009 foram definidas as suas características que diferenciariam do modelo relacional, tais como: (1) não utilizar *SQL* como linguagem, (2) geralmente serem projetos de código aberto, (3) necessidade de execução em *clusters*[8], (4) bancos que atendem a necessidade da *web* em pleno século XXI[9], (5) atuam sem esquema[10].

De acordo com [Elsmari e Navathe 2018] o termo *NoSQL* ou *Not Only SQL*, tem a finalidade de transmitir a ideia que muitas aplicações precisam de sistemas diferentes de *SQL* e a criação desse tipo de banco de dados se dá por meio da necessidade de gerenciar grandes quantidades de dados. Exemplos de organizações que se deparam com a necessidade de gerenciar grandes volumes de dados são Google, Yahoo, Amazon, Facebook e Twitter. Enfatiza-se que, a maioria dos *NoSQL* são bancos distribuídos, com foco no armazenamento semiestruturado, alto desempenho, disponibilidade, replicação dos dados e escalabilidade. Também é importante ressaltar que, conforme diversos autores e estudiosos na área de banco de dados, o objetivo deste modelo apresentado não é eliminar os bancos de dados relacionais,

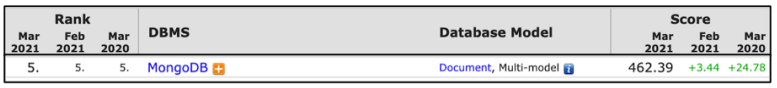
[8] Em banco de dados, *cluster* pode ser definido como um sistema onde dois ou mais computadores trabalham de maneira conjunta para realizar processamentos pesados, dividindo as tarefas de processamento e trabalhando como se fossem um único computador.

[9] Pode ser citado como um dos grandes problemas deste século, aplicações *web* que precisam operar com gigantescas cargas de dados. Assim, os bancos de dados do tipo não-relacional trazem o trabalho em escala para facilitar este processo.

[10] Esquema em banco de dados é sua estrutura descrita em uma linguagem formal suportada por um SGBD, ou seja, é a organização dos dados em tabelas no modelo relacional.

mas oferecer uma alternativa para lidar com o crescente número de informações que são gerenciadas e que exigem respostas em um curto tempo tanto no envio como no recebimento de respostas.

Quanto ao *ranking* dos bancos de dados [DBENGINES 2021] e conforme apresenta a Figura 10, em quinto lugar no mês de março de 2021, aparece *MongoDB* que é um exemplo de banco de dados do modelo não-relacional, apresentando um crescimento quanto a sua classificação.



Rank			DBMS	Database Model	Score		
Mar 2021	Feb 2021	Mar 2020			Mar 2021	Feb 2021	Mar 2020
5.	5.	5.	MongoDB	Document, Multi-model	462.39	+3.44	+24.78

Figura 10. *Ranking* com foco no banco de dados não-relacional [DBENGINES 2021]

2.1.2.1. Banco de dados não-relacional baseado em documentos

Os bancos de dados *NoSQL* são caracterizados por quatro principais categorias, sendo: baseado em documentos, armazenamento chave-valor, baseado em coluna e baseado em grafos [Elsmari e Navathe 2018]. Para este trabalho será abordado o banco de dados não-relacional baseado em documentos.

Estes sistemas armazenam dados como coleção de documentos, usando o formato *JSON* (*JavaScript Object Notation*), a qual possui um *id* para identificar os documentos. Possuem poderosas linguagens de consulta e podem ser facilmente escalados horizontalmente para acomodar grandes volumes de dados.

Como apresentado no *site* da [DevMedia 2011], comparando-se um banco de dados relacional com um banco orientado a documentos, pode-se dizer que o segundo não fornece relacionamento, ou seja, não há esquemas. Assim, ao invés dos dados serem armazenados em uma área de armazenamento separado, os bancos do tipo documentos integram esses dados ao próprio documento. Diante deste contexto, pode se dizer que uma das principais diferenças entre as estruturas, é que o banco orientado a documentos lida com documentos e não com registros como é o caso do modelo relacional onde são definidas tabelas e que são representadas através de linhas e colunas.

Nesse modelo há um conjunto de documentos e em cada documento existe um conjunto de campos (chaves) e o valor deste campo. Ele também não depende de um esquema rígido, ou seja, não exige uma estrutura fixa como ocorre nos bancos relacionais. Assim, é possível que ocorra uma atualização na estrutura do documento, com a adição de novos campos, por exemplo, sem causar qualquer anomalia ao banco de dados. Além disso, os tipos dos valores podem ser diferentes para cada registro, as colunas podem ter mais de um valor (*arrays*) e os registros podem ter uma estrutura aninhada. “Ao contrário dos bancos relacionais, no MongoDB não é necessário construir a estrutura do seu banco previamente antes de sair utilizando ele. Tudo é criado na medida que for usando[...].” [DUARTE JÚNIOR 2021].

Há vários tipos *NoSQL* baseados em documentos, sendo um dos mais populares o *MongoDB*, criado em 2007. É uma ferramenta *opensource* licenciada pela *GNU AGPL* (*Affero General Public License*).

A Tabela 1 apresenta uma comparação de terminologias entre *SQL* e *NoSQL MongoDB*.

Tabela 1: Terminologias entre *SQL* e *MongoDB*

SQL	***MongoDB***
Database	*Database*
Tabela	Coleção
Linha	Documento
Coluna	Campo
Chave primária	*ObjectId*
Join	Documentos embutidos ou referências

A Figura 11 apresenta uma comparação de conceitos associados ao banco de dados do *MongoDB* com um banco de dados relacional.

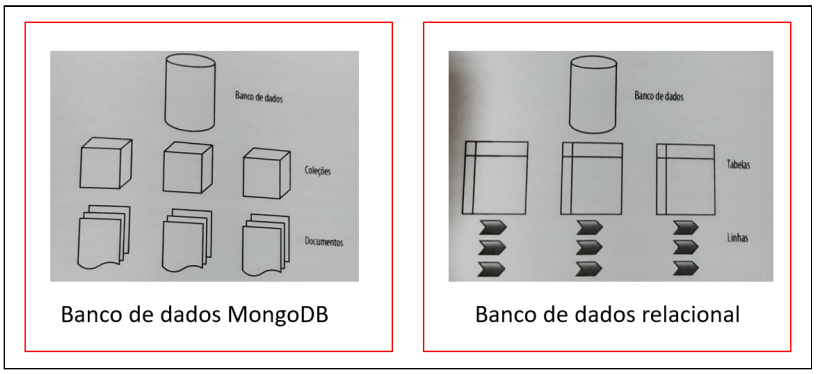


Figura 11. Comparação entre os modelos de banco de dados [HOWS, MEMBREY, PLUGGE 2015]

2.1.2.1.1. Comando para definição de coleção usando *NoSQL MongoDB*

Como já apresentado e reforçando com os autores [Hows, Membrey e Plugge 2015], o *MongoDB* não é relacional e nem tem esquemas, sendo assim, não está associado a nenhuma coluna ou tipo de dado predefinidos, como ocorre com bancos de dados relacionais. Todavia, independente do tipo de banco de dados, cria-se um *database* para armazenar a estrutura de tabelas ou coleções de uma base. Os autores [Hows, Membrey e Plugge 2015] também enfatizam que o *MongoDB* armazena dados em formato de documento e esse pode ser comparado a uma linha do *SQL*.

Em *SQL* para se criar um *database*[11] é usada a instrução "*create database* <nome_base>". Em *NoSQL* também se usa este termo e conceito e a instrução é "*use* <nome_base>".

[11] Antes de se criar uma tabela (*table* - no modelo relacional) ou uma coleção (*collection* - no modelo não-relacional) , é necessário criar um *database* que, tem como objetivo, definir um local onde ficarão armazenados os dados. A terminologia *database* pode ser comparada como uma pasta em um sistema operacional, que tem como objetivo armazenar dados com o propósito específico da sua criação.

Quanto a operação responsável pela criação das coleções de um banco orientado a documento é denominado *createCollection* [ELMASRI NAVATHE 2018]. Na Figura 12, é apresentado um exemplo da operação.

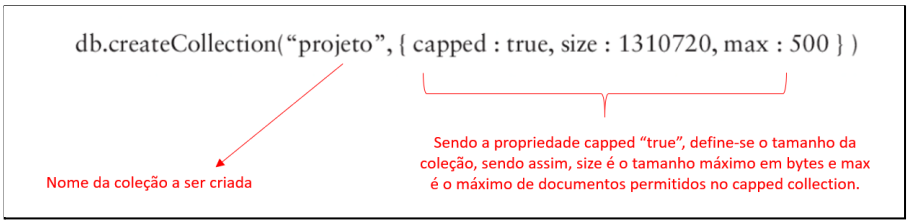


Figura 12. Exemplo de criação de *collection*

2.1.2.1.2. Comandos para inserção de documento através da coleção

Considerando o *collection* criado e representado na Figura 12, para definir um documento na coleção "projeto", o *MongoDB* possui a operação *insert.* A Figura 13 traz um exemplo dessa opção enfatizando que entre os caracteres de chave, a ordem se dá pelo nome do campo, seguido pelo caractere dois pontos. Entre aspas está o conteúdo a ser armazenado no campo correspondente [ELMASRI NAVATHE 2018].

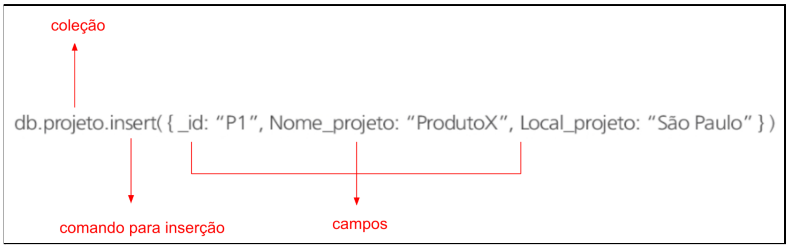


Figura 13. Exemplo de inserção na *collection*

2.1.2.1.3. Comandos para consulta de documento no *MongoDB*

Consultas no modelo não-relacional podem ser equivalentes com as instruções do modelo relacional, podendo estas ser simples ou com o uso de operadores relacionais e lógicos [HOWS, MEMBREY E PLUGGE 2015].

No MongoDB uma simples consulta é caracterizada com o uso de chaves e valores através do uso do método find().

A Figura 14 apresenta um exemplo de consulta simples, considerando a inserção representada na Figura 13.

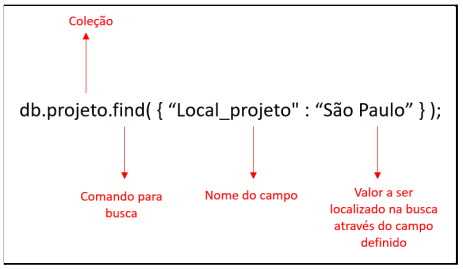


Figura 14. Exemplo de consulta simples

Para efeito de informação e para descrever um exemplo de consulta avançada (Figura 15), o *MongoDB* utiliza os seguintes operadores: $gte (operador relacional "maior ou igual"), $lte (operador relacional "menor ou igual"), $min (operador relacional "menor"), $max (operador relacional "maior"), $and (operador lógico "and") e $or (operador lógico "or")[12].

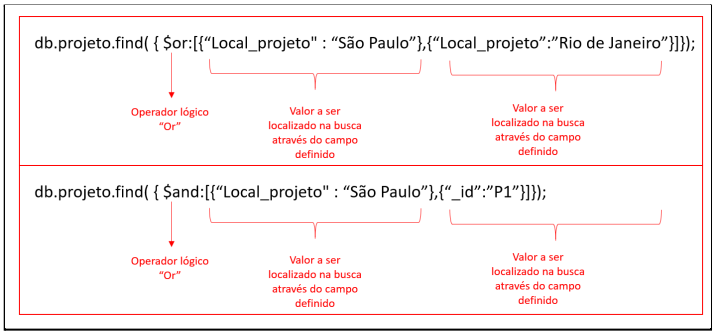


Figura 15. Exemplo de consulta avançada

Considerando que a Figura 15 possui dois exemplos de consulta avançada, a primeira busca utilizando o operador lógico "$or", retornará documentos que possui o campo "Local_projeto" com o conteúdo "São Paulo" ou "Rio de Janeiro". Já a segunda busca utiliza o operador lógico "$and", tendo como objetivo retornar documentos que atendam as duas condições, ou seja, o campo "Local_projeto" com o conteúdo "São Paulo" e " _id" com o conteúdo "P1".

2.1.2.1.4. Relacionar dados no *MongoDB*

Como já apresentado, o *MongoDB* não possui um esquema pré-definido como os bancos *SQ*L onde é necessário declarar uma estrutura de tabela para que, após, possam ser inseridas as tuplas. Sendo assim, outro diferencial na modelagem é que o banco de dados não-relacional não usa *JOINs* entre tabelas. Alguns autores destacam em suas obras que no modelo não-relacional devem ser evitados relacionamentos, todavia sendo necessários então fundamenta-se o uso de *SQL*. Como cita [Duarte Júnior 2021], "não utilizar *MongoDB* quando relacionamentos entre diversas entidades são importantes para o seu sistema. Se for ter de usar muitas 'chaves estrangeiras' e '*JOINs*' estará usando do jeito errado, ou, ao menos, não do jeito mais indicado."

Como não há uso de chaves para relacionamentos entre tabelas, o novo paradigma para se trabalhar com banco de dados não-relacional que, nessa proposta é o *MongoDB*, traz duas possibilidades de se criar essas relações, sendo:

- Por referências de documentos: de uma coleção em outras. É chamado de relacionamento lógico e segue um modelo de projeto normalizado.
- Por documentos embutidos: é como um sub-documento que armazena todas as referências em formato de *arrays* ou campos do documento. As relações estão associadas a um modelo desnormalizado.

Os autores [Elsmari e Navathe 2018] destacam que "o usuário pode escolher um projeto normalizado (semelhante às tuplas relacionais normalizadas) ou um projeto

[12] Para estudo dos operadores relacionais e lógicos, considerando a sintaxe de utilização através da instrução find no MongoDB foi utilizado o link http://leandroflora-001-site2.htempurl.com/2019/01/26/consultas-mongodb/.

desnormalizado (semelhante a documentos *XML* ou objetos complexos)". Na Figura 16 é apresentado um documento normalizado em *MongoDB*, onde a coleção projeto possui o campo *_id* que é relacionado com a coleção trabalhador através do campo *IdProjeto*.

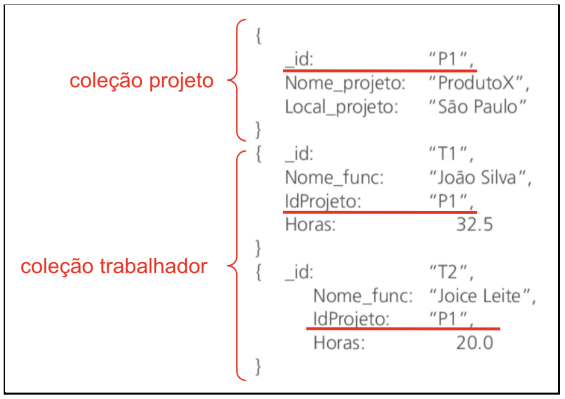


Figura 16. Exemplo de documentos normalizados no MongoDB

Outra opção de relacionamento no *MongoDB* é chamado de documento desnormalizado, como é representado na Figura 17. Nesse exemplo, toda a estrutura se refere a uma única coleção onde os campos que poderiam ser correspondentes a outra coleção são embutidos dentro da mesma *collection*, denominada subcoleção. Sendo assim, considernando o exemplo, os Trabalhadores são listados dentro de um vetor que especifica o campo "Nome_func" e valor do campo "Horas".

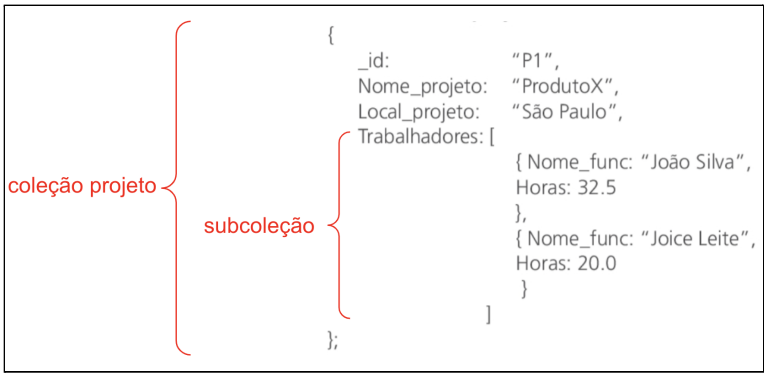


Figura 17. Exemplo de documentos desnormalizados no *MongoDB*

3. Metodologia e Materiais

Essa seção descreve as metodologias e materiais utilizados para o desenvolvimento do presente trabalho.

Primeiramente foi realizada uma pesquisa bibliográfica sobre banco de dados relacional e não-relacional, considerando suas características e definições. Logo após uma pesquisa sobre aplicações que executam migrações para o modelo não-relacional, métodos de migrações e propostas de migrações. Baseado no *ranking* do *site* [DBEngines 2021], foram selecionados dois bancos de dados *open-source*, sendo um relacional (*MySQL* - na relação do

site se apresenta em segundo lugar) e o outro não-relacional (*MongoDB* - na relação do *site* se apresenta em quinto lugar).

Para aplicar as ferramentas e métodos de migração estudados, foi utilizado como estudo de caso o Diagrama de Entidade-Relacionamento (DER) do Trabalho de Conclusão de Curso (TCC) do autor [Pereira 2015] que propõe o aplicativo intulado "IFSP Serviços de smartphone para uso de docentes e discentes de uma instituição de ensino".

Quanto aos materiais utilizados para migrações foram utilizadas as ferramentas *MongoDB* Compass, PHPMyAdmin e a ferramenta Studio 3T.

4. Desenvolvimento

Com o levantamento bibliográfico, o levantamento de requisitos e considerando o estudo de caso baseado no DER [PEREIRA 2015], foram selecionados dois bancos de dados, sendo o *MySQL* como banco de dados relacional e o *MongoDB* como banco de dados não relacional. A escolha destes dois diferentes paradigmas na área de banco de dados se dá para propor os objetivos da proposta.

A Figura 18 [PEREIRA 2015] apresenta o DER, estando representado através da ferramenta PHPMyAdmin. Essa base de dados foi criada no banco de dados relacional com o uso da linguagem *SQL* através das instruções apresentadas na subseção 2.1.1.1.1.

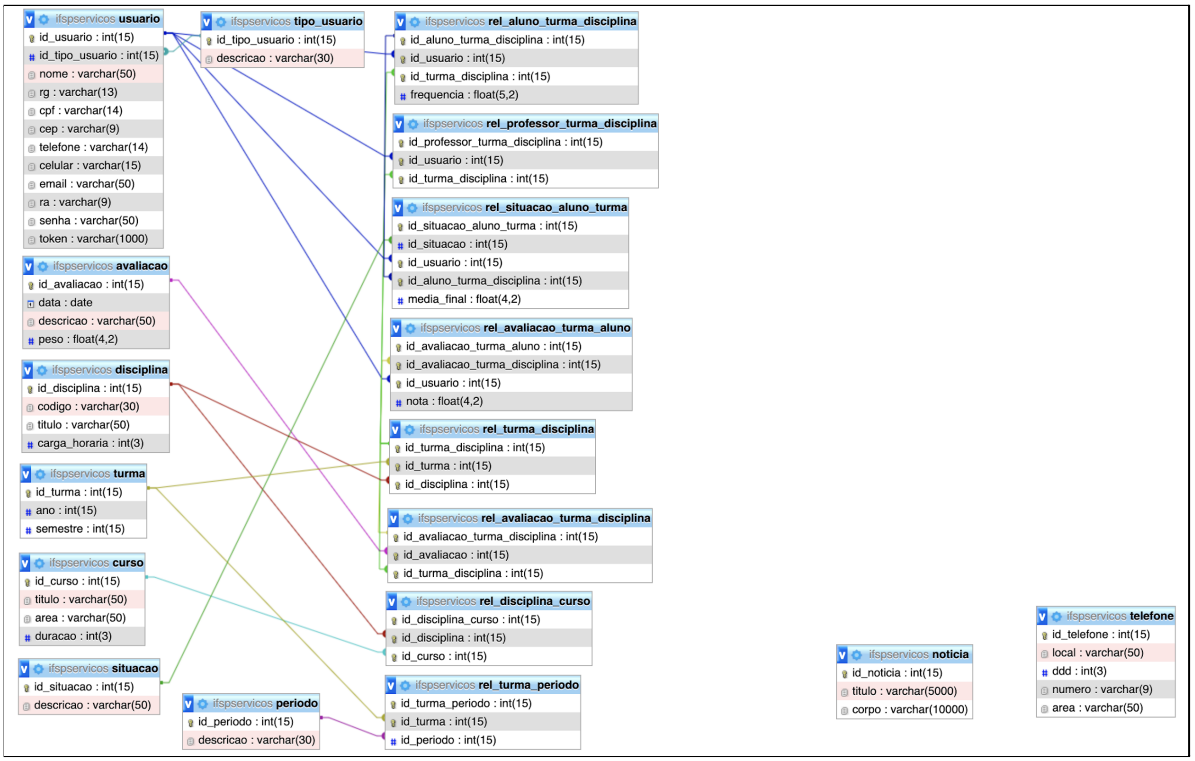


Figura 18. DER baseado no TCC do autor [Pereira 2015]

Diante do modelo representado na Figura 18, serão apresentadas três propostas de migração de um banco de dados *SQL->MySQL* para *NoSQL->MongoDB*, abordando cada uma delas em uma subseção. A primeira proposta consiste em apresentar uma ferramenta paga disponível no mercado, a segunda proposta é converter uma base em uma linguagem que seja reconhecida no modelo não-relacional e, por fim, a terceira proposta se baseia na

construção de uma base em *NoSQL* através de comandos, conforme exposto na subseção 2.1.2.1.

4.1. Studio 3T

O aplicativo Studio 3T oferece ferramentas pagas que auxiliam desenvolvedores e administradores do *MongoDB* em migração de bases, tendo compatibilidade nos sistemas operacionais Windows, Mac e Linux, porém a versão analisada possui licença *shareware*. A ferramenta contém uma interface amigável, possibilitando usuários com pouco conhecimento na ferramenta em manuseá-la com facilidade, além da possibilidade de acesso simultâneo com várias bases de dados e com operações executadas em tempo real.

Os primeiros passos após efetuar o *download* são a configuração dos bancos de dados de origem e destino[13]. Nesta proposta, a base utilizada é do estudo de caso apresentado na Figura 18, sendo em *MySQL*. A Figura 19 retrata a configuração da base origem.

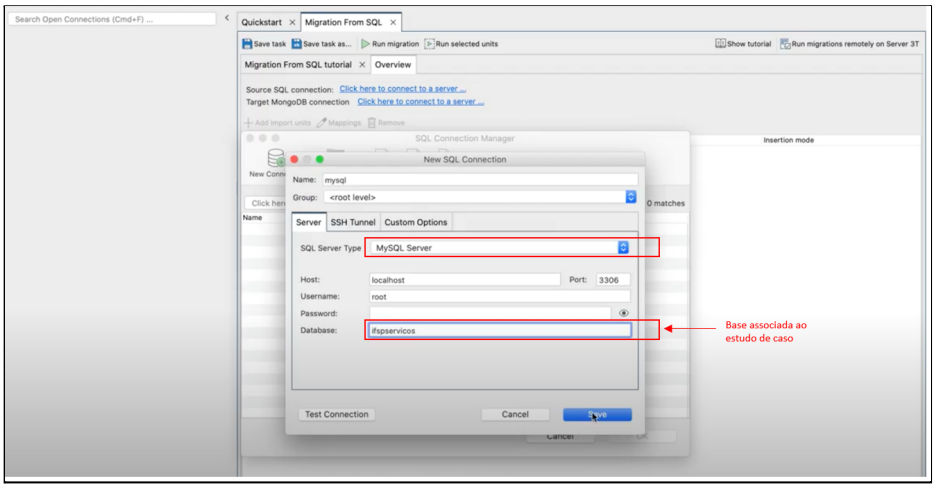


Figura 19. Configuração da base de origem

Com a configuração da base origem, como é apresentada na Figura 20, define-se a base destino, ou seja, o modelo para migração. Diante da proposta, seleciona-se *NoSQL*. Para esta ação foi criada uma conta gratuita no *cluster* do *MongoDB*.

[13] O processo de configuração pode ser observado através do vídeo pelo *link* https://drive.google.com/file/d/1uvz2jloS_nAA_Qz5at4EPMjKfQp5UYwi/view?usp=sharing.

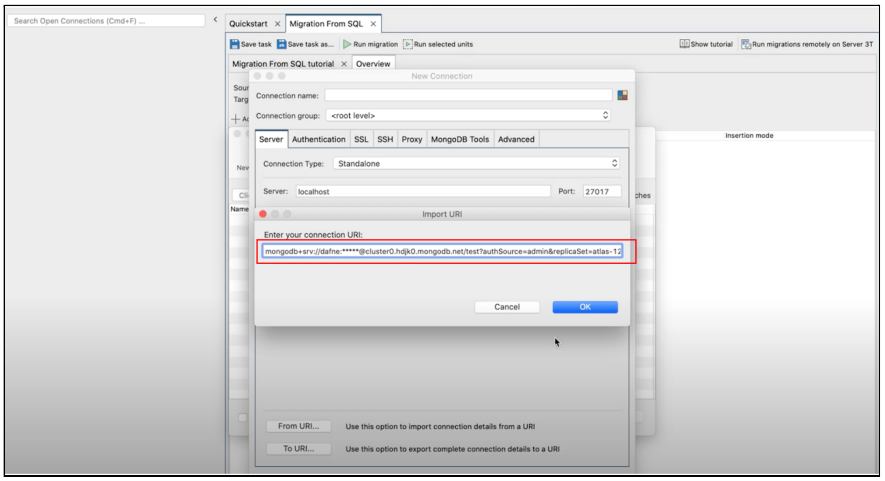


Figura 20. Configuração da base de destino

Após as ações apresentadas, o próximo passo é executar a migração, sendo que, para esta proposta, foram observados dois cenários com o objetivo de analisar o comportamento e apontar os resultados obtidos.

- No cenário 1[14]: foi utilizada a base de dados do estudo de caso vazia, sem dados inseridos no banco, apenas com as estruturas de tabelas e relacionamentos. Após a migração, foram listadas as tabelas em uma pasta denominada Collections. Essa pasta, contém todas as tabelas do banco de dados relacional que foram migradas. Na Figura 21, são exibidos os atributos da tabela "avaliacao", como exemplo, que podem ser visualizados em modo tabela ou *JSON*. Identificou-se que a estrutura da tabela contém apenas o *_id* como atributo. O teste com uma base vazia resultou em uma migração incorreta.

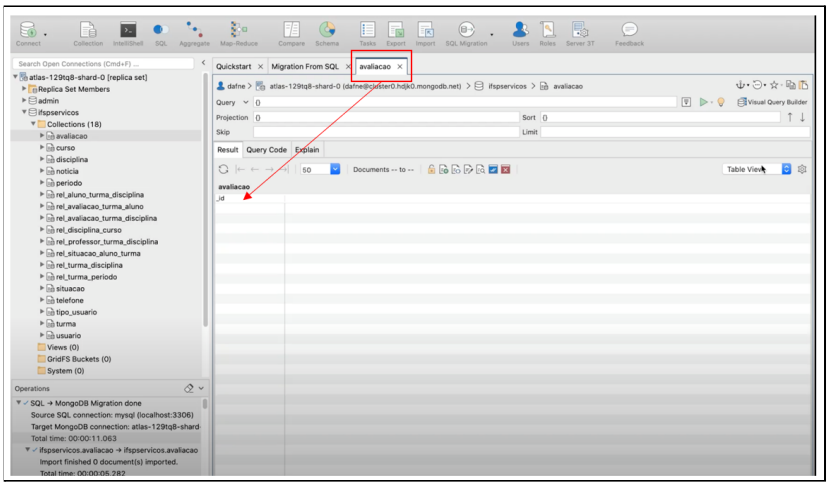


Figura 21. Representação do cenário 1

[14] A migração pela ferramenta Studio 3T sem dados pode ser observada através do vídeo pelo *link* https://drive.google.com/file/d/1PpyXUCVthy_whyJCyj4fYo4n4kdLOJqI/view?usp=sharing.

- No cenário 2[15]: foi utilizada a base de dados do estudo de caso com dados inseridos nas tabelas. Na Figura 22 pode ser observada a migração da tabela avaliação. Observa-se que, diferente do cenário 1, o processo foi concluído com êxito.

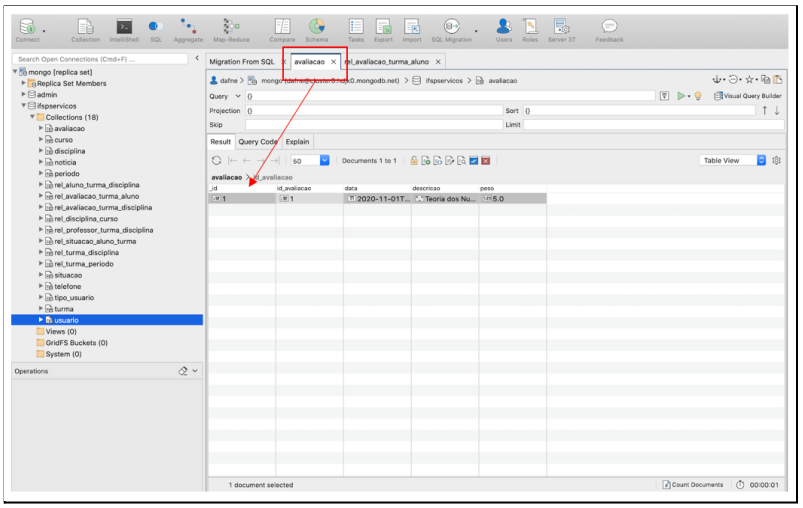


Figura 22. Representação do cenário 2

Outro ponto observado nesse cenário, foi referente às tabelas que possuem relacionamentos no banco de dados relacional. O conteúdo do campo equivale ao valor que está em outra *collection*, mas não estão associadas. Isso acaba dificultando em identificar se o campo é chave estrangeira, principalmente nas collections que há mais de duas chaves estrangeiras oriundas do modelo relacional. Apenas enfatizando, conforme exposto no referencial teórico, que no modelo não-relacional não há conceitos de campos do tipo chave primária e chave estrangeira.

4.2. Importação *JSON*

A segunda proposta de migração está baseado na conversão da base de dados do *MySQL* em uma estrutura *JSON*, já que no *MongoDB* a estrutura dos documentos é em *JSON*[16]. Conforme apresenta a Figura 23, essa funcionalidade está disponível no phpMyAdmin, através do menu Exportar, opção de formato *JSON*. É importante enfatizar que, para testar esta proposta de migração está sendo utilizada a base do estudo de caso apresentado (ifspservicos).

[15] A migração pela ferramenta Studio 3T com dados pode ser observada através do vídeo pelo *link* https://drive.google.com/file/d/1aUFPulXHnpNDOqdOq9gahsAivsFv0-GU/view?usp=sharing.

[16] A migração com o uso do PHPMyAdmin de *SQL* para *JSON* pode ser observada através do vídeo pelo *link* https://drive.google.com/file/d/1xD9AnZ8p1moGlUfDB5YZvseFAkU3y1bP/view?usp=sharing.

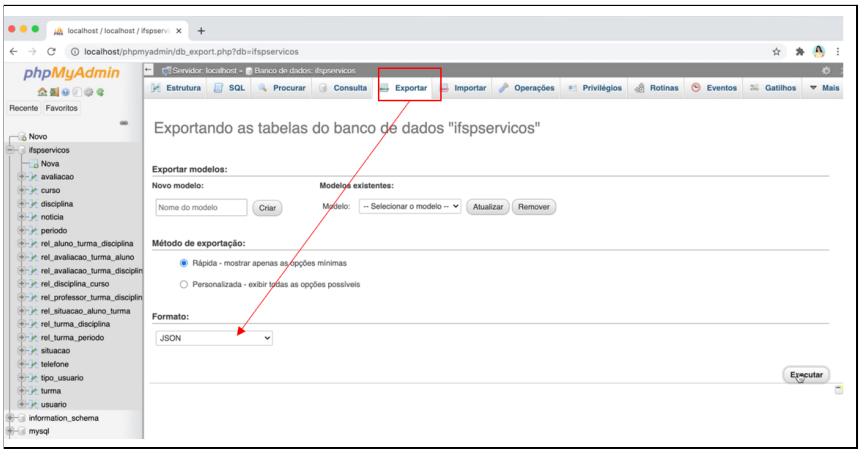


Figura 23. Exportar do *SQL* para *JSON* através do phpMyAdmin

Após a migração, o próximo passo foi a criação de uma base de dados no *MongoDB* e uma *collection* para importar o arquivo convertido. Como apresenta a Figura 24, observou-se que toda a base ficou armazenada em uma única *collection*, podendo facilitar a performance nas buscas, já que todos os documentos estão armazenados em uma única coleção. Enfatiza-se que cada coleção está representada por um documento e cada documento contêm campos embutidos referente aos campos que cada tabela do *MySQL* armazena os dados.

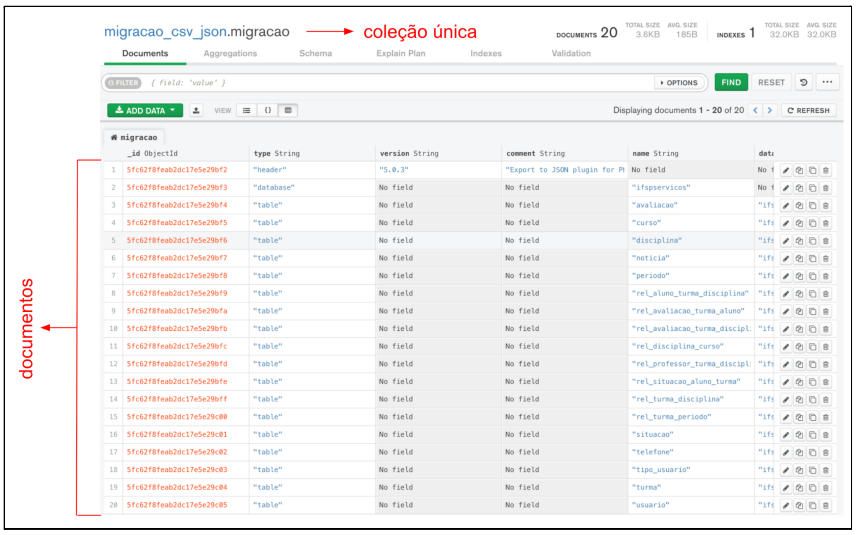


Figura 24. Estrutura criada no MongoDB através da migração em JSON

Como está tudo unificado em uma única *collection* a busca de um elemento específico de um documento, resultará numa sintaxe um pouco diferente. Nessa estrutura os dados ficam armazenados dentro de um *array*, ou seja, todo documento haverá uma *subcollection* que é um *array* com os dados do que era uma tabela. Na busca desse elemento, terá que mencionar no comando **db.collection.find({"<nome_array.campo":"valor_pesquisado"})**, como exemplifica a Figura 25.

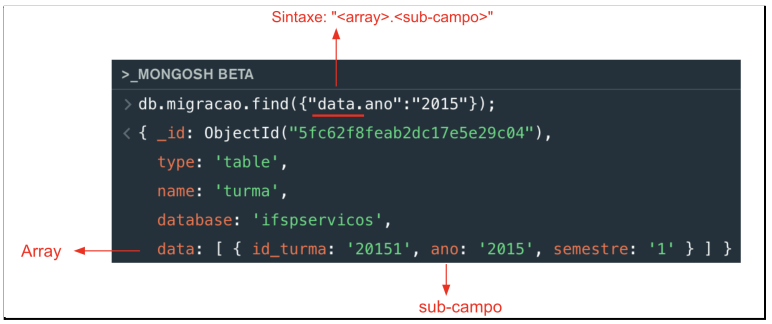


Figura 25. Exemplo de busca em uma collection no *MongoDB*

4.3. Migração de uma base relacional para não-relacional através de comandos

A terceira proposta para migração do banco de dados relacional para não-relacional, se dá no formato manual com execução das instruções no *MongoDB*. Baseado no DER do autor [Pereira 2015] são criadas as *collections* principais que formam a base de dados, tendo como objetivo detalhar as decisões tomadas na transição entre o processo de migração.

Para isso duas perguntas foram pensadas no início da transição manual, a fim de entender como a modelagem final do banco *NoSQL* se tornaria: 1. Todas as tabelas serão *collections*? 2. Como é realizado o relacionamento das *collections*?

Diante desta análise inicial, foram definidos os seguintes passos:

- Criação do *database*: é o ponto de partida para iniciar a migração. O comando *use* substitui o *create database* utilizado nos bancos de dados *MySQL* para criação da base de dados, conforme exibido na Figura 26.

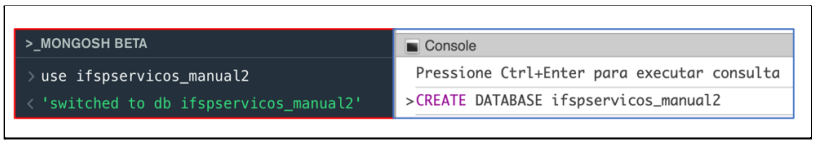


Figura 26. Criação do *database* no *MongoDB* X *MySQL*

- Criação das *collections*: as chamadas coleções no *MongoDB* são comparadas com as tabelas do *MySQL*, todavia segue outro paradigma quanto a criação. A definição da *collection* adiciona os dados que serão armazenados diretamente em cada campo, dispensando a necessidade de criar uma estrutura física para indicar o tipo de dado que será armazenado como ocorre nos bancos de dados relacionais. Para abordagem do estudo de caso é necessária a criação de várias *collections*, porém para exemplificar neste artigo será abordada a criação da *collection* "curso", listando as demais no [Anexo I] deste documento. A Figura 27 traz a criação da collection, fazendo uma comparação com a criação da tabela no modelo relacional.

Figura 27. Criação da *collection* no *MongoDB* X tabela no *MySQL*

Como já apresentado no referencial teórico, ressalta-se que todas as *collections* já possui um campo *_id* que o próprio *MongoDB* cria automaticamente. Esse campo *_id* é um identificador único para o documento. Então, não há a necessidade de criar um campo para armazenar esse identificador como é criado nos bancos de dados relacionais no formato de *primary key*.

- Definições das relações (embutido ou referencial): como já abordado no referencial teórico, o relacionamento no *MongoDB* pode ser realizado de duas formas: embutido (não normalizado) ou referencial (normalizado). O embutido é utilizado quando os dados são incluídos no mesmo documento, já o referencial incorpora no documento algum dado que já foi armazenado por outra coleção separada. Cria-se uma espécie de *link* entre a campo A da collectionA e o campo B da collectionB, por exemplo.

Para este trabalho, foi aplicada a forma referencial que, para exemplificar, a Figura 28 traz essa relação entre as *collections* "usuario" e "tipo_usuario"[17]. Ressalta-se que a referência entre os campos é realizada a partir do _id ("ObjectId"), então a *collection* "tipo_usuario" precisa conter um documento armazenado e assim utilizar o _id do documento para referenciar a *collection* "usuario".

Figura 28. Relacionamento referencial no *MongoDB*

- Consulta de documentos: a busca por dados no *MongoDB* e *MySQL* são semelhantes. O comando *select* utilizado para consultas em bancos de dados relacionais é substituído por *find*. No exemplo da Figura 29 (a) contém uma busca ampla que retorna todos os documentos que estão armazenados na *collection* curso; (b) contém uma busca mais detalhada adicionando um parâmetro, somente documentos que

[17] O relacionamento referencial adotado pode ser observado através do vídeo pelo *link* https://drive.google.com/file/d/1lxx9Axh8ZVx6Co3HHZDZjpwiWjc7i7j8/view?usp=sharing.

atendem ao parâmetro especificado será retornado como resultado da busca.

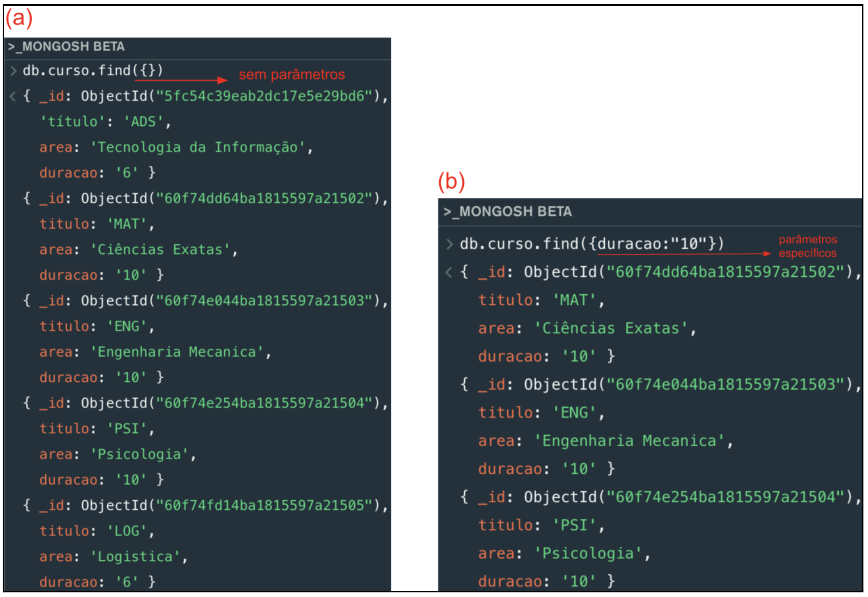


Figura 29. Exemplos de consulta no MongoDB

5. Resultados

Após o término dos testes aplicados nas três propostas para migração de uma base relacional para não-relacional, foram observadas vantagens e desvantagens, sendo descritas nos próximos parágrafos.

Proposta 1: foi utilizada a ferramenta Studio 3T. Considerando a base do estudo de caso, a qual continha a estrutura das tabelas e os dados inseridos nas mesmas, o modelo estrutural não sofreu grandes modificações ao migrar do relacional para o *NoSQL*, mas foi observado que no *MongoDB* foi criado um atributo em cada *collection* denominado "ObjectId". Em um segundo teste com a base de dados sem tuplas, o *MongoDB* não criou os atributos de cada *collection*, porém como o intuito deste trabalho foi a migração com dados, este segundo teste não impactou o resultado final, destacando esta observação apenas para efeito de informação.

Proposta 2: utilizou-se uma importação *JSON* que é a linguagem nativa do *MongoDB* para migrar os dados. Para essa migração pode ser usado o phpMyAdmin que é um gerenciador de banco de dados em *MySQL* onde possui uma funcionalidade nativa de exportar o banco em formato *JSON*. A desvantagem dessa proposta é a necessidade de criar um *database* e *collection* manualmente no *MongoDB* para fazer a importação. Um ponto que pode ser considerado como vantagem, mas ao mesmo tempo desvantagem, é a concentração de toda a base em uma única *collection*, pois é mais eficaz uma busca em uma única *collection* que contém todas as informações ao invés de buscar em várias *collections*. Todavia, se analisar uma validação de acesso ao login, será retornado toda a *collection* e isso pode haver uma falha de segurança, pois está retornando outros dados que não são necessários para essa ação.

Proposta 3: criou uma base e toda estrutura no *NoSQL*, sendo construída manualmente através dos comandos abordados no referencial teórico e como foi aplicado o relacionamento referencial, o resultado da quantidade de *collections* no *MongoDB* foi igual a quantidade de tabelas no *MySQL*. Outra vantagem que envolve o desenvolvimento é sobre a facilidade de criar uma *collection* sem a necessidade de declarar o tipo de dado que será armazenado. A desvantagem é que ao criar um relacionamento referencial, a *collection* cria um novo campo que está utilizando o mesmo valor de outra *collection*, podendo ser considerado que há redundância de dados.

6. Conclusões

Após os testes, conclui-se que a primeira proposta traz como vantagem a automatização de todo o processo de migração, porém é fundamental a parametrização correta das etapas, além dos resultados finais apresentar a diferença na escolha de uma base vazia e com dados, conforme exposto no primeiro cenário da Seção 5. Um ponto negativo sobre a ferramenta para migração é que não há no mercado uma versão gratuita, apenas *trial* com a permissão para quinze dias de teste, sendo que após este prazo é necessário pagar uma licença para continuar com os testes de migração.

Na segunda proposta a vantagem é dispensar a necessidade de pré-configurações entre os servidores, conforme a ferramenta Studio 3T apresenta na proposta 1. Usando um SGBD relacional é possível fazer uma exportação da base já no formato *JSON*, que é um modelo compatível para o *NoSQL*.

Por fim, a terceira proposta traz como aspecto positivo o aprendizado de um tipo de banco diferente do modelo que é proposto no Curso Superior, todavia o processo é árduo, caso o modelo relacional seja extenso para fazer todo o processo de migração, considerando as normas do modelo *NoSQL*. Assim, propõe uma ferramenta para automatizar este processo, sendo exposto na Seção 6.

Para desenvolvimento deste trabalho e até mesmo destacando as competências e habilidades adquiridas no decorrer do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, houve aplicação de conceitos associados às disciplinas de Algoritmo e Lógica de Programação, Banco de Dados, Engenharia de *Software*, além da Metodologia de Pesquisa e Projeto de Sistemas para planejamento e desenvolvimento teórico-técnico.

6. Trabalhos Futuros

Baseando-se na terceira proposta apresentada no presente artigo, propõe-se a implementação de um aplicativo que faça o processo de migração, considerando a escolha de uma base relacional para ser migrada para uma base não-relacional, conforme é sugerido no diagrama de fluxo apresentado na Figura 30. Os passos para uma futura implementação estão enumerados na ordem das funcionalidades.

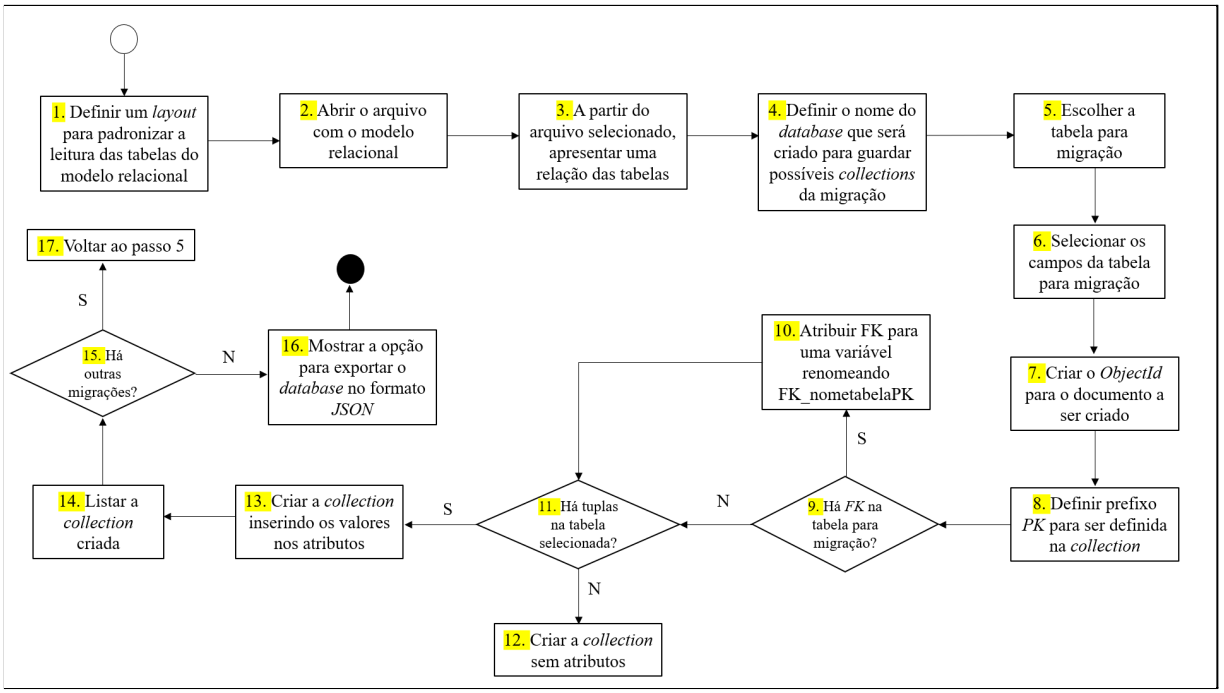


Figura 30. Diagrama de fluxo para desenvolvimento de um aplicativo para migração

Referências

DBEngines. “DB-Engines Ranking”. https://db-engines.com/en/ranking. [Online: acessado em 10 de fevereiro de 2021].

DevMedia. “Big Data Tutorial”. https://www.devmedia.com.br/big-data-tutorial/30918. [Online: acessado em 15 de agosto de 2020].

Duarte Júnior, L. “MongoDB para Iniciantes”. https://www.pensecomigo.com.br/livro-mongodb-para-iniciantes-pdf-luiz-duarte/. [Online: acessado em 20 de fevereiro de 2021].

Elmasri, R., Navathe, S. B. (2018). “Sistemas de Banco de Dados”. Editora Pearson, 7ª Edição.

Hows, D., Membrey, P., Plugge, E. (2015). “Introdução ao MongoDB”. Editora Novatec.

Heuser, C. A. (2009). “Projeto de Banco de Dados”. Editora Bookman, 6ª Edição.

Martin, F., Pramod J. S. (2013). “NoSQL Essencial: um Guia Conciso Para o Mundo Emergente da Persistência Poliglota”. Novatec.

Pereira, G. L. (2015). “IFSP Serviços: um aplicativo para acesso aos dados da vida acadêmica discente”. https://hto.ifsp.edu.br/portal/images/thumbnails/images/IFSP/Cursos/Coord_ADS/Arquivos/TCCs/2015/TCC_GuilhermeLimaPereira.pdf. [Online: acessado em 10 janeiro de 2020].

Sommerville, I. (2019). “Engenharia de Software”. Editora Pearson, 10ª Edição.

Anexo I

```
>  CREATE TABLE telefone(
       id_telefone int(15) AUTO_INCREMENT PRIMARY KEY,
       local       varchar(50) NOT NULL,
       ddd         int(3) NOT NULL,
       numero      varchar(9) NOT NULL,
       area        varchar(50) NOT NULL
   );
> db.telefone.insertOne({local:"Hortolandia",ddd:"19",numero:"999999999",area:"Interior-SP"})
```

```
>  CREATE TABLE avaliacao(
     id_avaliacao int(15) AUTO_INCREMENT PRIMARY KEY,
     data         date        NOT NULL,
     descricao    varchar(50) NOT NULL,
     peso         float(4,2)  NOT NULL
   );
> db.avaliacao.insertOne({data:"15/02/2015", descricao: "Matematica Basica",peso:"10"})
```

```
>  CREATE TABLE curso(
     id_curso int(15) AUTO_INCREMENT PRIMARY KEY,
     titulo   varchar(50) NOT NULL,
     area     varchar(50) NOT NULL,
     duracao  int(3) #Em semestres
   );
> db.curso.insertOne({titulo:"Engenharia Agronoma", area: "Exatas",duracao:"10"})
```

```
>  CREATE TABLE disciplina(
     id_disciplina int(15) AUTO_INCREMENT PRIMARY KEY,
     codigo        varchar(30) NOT NULL, #Alternate key
     titulo        varchar(50) NOT NULL,
     carga_horaria int(3)                #Em horas
   );
> db.disciplina.insertOne({codigo:"ED1", titulo:"Estrutura de Dados I", cargahoraria:"80"})
```

```
>  CREATE TABLE periodo(
     id_periodo int(15) AUTO_INCREMENT PRIMARY KEY,
     descricao  varchar(30) NOT NULL
   );
> db.periodo.insertOne({descricao:"MATUTINO"})
```

```
>  CREATE TABLE situacao(
     id_situacao int(15) AUTO_INCREMENT PRIMARY KEY,
     descricao   varchar(50)  #Aprovado, Reprovado, Em Curso
   );
> db.situacao.insertOne({descricao:"REPROVADO"})
```

```
>  CREATE TABLE tipo_usuario(
       id_tipo_usuario int(15) AUTO_INCREMENT PRIMARY KEY,
       descricao       varchar(30) NOT NULL
   );
> db.tipo_usuario.insertOne({descricao:"ADMINISTRATIVO"})
```

```
>  CREATE TABLE turma(
     id_turma int(15) AUTO_INCREMENT PRIMARY KEY,
     ano      int(15) NOT NULL,
     semestre int(15) NOT NULL
   );
> db.turma.insertOne({ano:"2016",semestre:"1"})
```

```
>  CREATE TABLE usuario(
       id_usuario      int(15) AUTO_INCREMENT PRIMARY KEY,
       id_tipo_usuario int(15),
       nome            varchar(50) NOT NULL,
       rg              varchar(13),
       cpf             varchar(14) NOT NULL,
       cep             varchar(9)  NOT NULL,
       telefone        varchar(14),
       celular         varchar(15),
       email           varchar(50),
       ra              varchar(9)  NOT NULL,
       senha           varchar(50) NOT NULL,
       token           varchar(1000)
   );
> db.usuario.insertOne({nome:"João Santos",rg:"3333333333",cpf:"33333333333", cep:"13130000",
   telefone:"03333333333", celular:"33333333333", email:"joaosantos@ifsp.com",
   ra:"3000000",senha:"joao123", token:"ifsp123"
```

```
>  CREATE TABLE rel_aluno_turma_disciplina(
     id_aluno_turma_disciplina int(15)    AUTO_INCREMENT PRIMARY KEY,
     id_usuario                int(15)    NOT NULL,
     id_turma_disciplina       int(15)    NOT NULL,
     frequencia                float(5,2) NOT NULL,
     UNIQUE(id_usuario, id_turma_disciplina)
   );
> db.rel_aluno_turma_disciplina.insertOne({frequencia:"80.00"})
```

```
>  CREATE TABLE rel_avaliacao_turma_aluno(
     id_avaliacao_turma_aluno      int(15)     AUTO_INCREMENT PRIMARY KEY,
     id_avaliacao_turma_disciplina int(15)     NOT NULL,
     id_usuario                    int(15)     NOT NULL,
     nota                          float(4,2)  NOT NULL,
     UNIQUE(id_usuario, id_avaliacao_turma_disciplina)
   );
> db.rel_avaliacao_turma_aluno.insertOne({nota:"8.00"})
```

```
>  CREATE TABLE rel_avaliacao_turma_disciplina(
     id_avaliacao_turma_disciplina int(15) AUTO_INCREMENT PRIMARY KEY,
     id_avaliacao                  int(15) NOT NULL,
     id_turma_disciplina           int(15) NOT NULL,
     UNIQUE(id_avaliacao, id_turma_disciplina)
   );
> db.rel_avaliacao_turma_disciplina.insertOne({id_avaliacao: ObjectId("5fc54c9deab2dc17e5e29bd9")})
```

```
>  CREATE TABLE rel_disciplina_curso(
     id_disciplina_curso int(15) AUTO_INCREMENT PRIMARY KEY,
     id_disciplina       int(15) NOT NULL,
     id_curso            int(15) NOT NULL,
     UNIQUE(id_disciplina, id_curso)
   );
> db.rel_disciplina_curso.insertOne({id_disciplina: ObjectId('5fc54c6eeab2dc17e5e29bd8')})
```

```
> CREATE TABLE rel_situacao_aluno_turma(
    id_situacao_aluno_turma   int(15) AUTO_INCREMENT PRIMARY KEY,
    id_situacao               int(15) NOT NULL,
    id_usuario                int(15) NOT NULL,
    id_aluno_turma_disciplina int(15) NOT NULL,
    media_final               float(4,2)  NOT NULL,
    UNIQUE(id_usuario, id_aluno_turma_disciplina)
  );
```

```
> db.rel_situacao_aluno_turma.insertOne({media_final:"8.00"})
```

```
> CREATE TABLE rel_turma_disciplina(
    id_turma_disciplina int(15) AUTO_INCREMENT PRIMARY KEY,
    id_turma            int(15) NOT NULL,
    id_disciplina       int(15) NOT NULL,
    UNIQUE(id_turma, id_disciplina)
  );
```

```
> db.rel_turma_disciplina.insertOne({id_turma: ObjectId('5fc54bdceab2dc17e5e29bd5')})
```

```
> CREATE TABLE rel_turma_periodo(
    id_turma_periodo int(15) AUTO_INCREMENT PRIMARY KEY,
    id_turma         int(15) NOT NULL,
    id_periodo       int(15) NOT NULL,
    UNIQUE(id_turma)
  );
```

```
> db.rel_turma_periodo.insertOne({id_turma: ObjectId('5fc54bdceab2dc17e5e29bd5')})
```

```
CREATE TABLE rel_professor_turma_disciplina(
  id_professor_turma_disciplina int(15)  AUTO_INCREMENT PRIMARY KEY,
  id_usuario                    int(15)  NOT NULL,
  id_turma_disciplina           int(15)  NOT NULL,
  UNIQUE(id_usuario, id_turma_disciplina)
)
```

```
> db.rel_professor_turma_disciplina.insertOne({id_usuario: ObjectId('60f89c8e2ac2d5711ae8896a')})
```

Relacionamento

```
> ALTER TABLE usuario
    ADD CONSTRAINT FK_rel_usuario_tipo_usuario FOREIGN KEY (id_tipo_usuario) REFERENCES tipo_usuario(id_tipo_usuario);
```

```
> db.usuario.updateOne({cpf:"11111111111"},{$set:{tipo_usuario:ObjectId('5fc6c47eeab2dc17e5e29c06')}})
```

```
> ALTER TABLE rel_turma_periodo
    ADD CONSTRAINT FK_rel_turma_periodo         FOREIGN KEY (id_turma)   REFERENCES turma(id_turma),
    ADD CONSTRAINT FK_rel_turma_periodo_periodo FOREIGN KEY (id_periodo) REFERENCES periodo(id_periodo);
```

```
> db.rel_turma_periodo.updateOne({_id: ObjectId('60f8a9872ac2d5711ae8896d')},{$set:{id_periodo: ObjectId('5fc54c4feab2dc17e5e29bd7')}})
```

```
> ALTER TABLE rel_turma_disciplina
    ADD CONSTRAINT FK_rel_turma_disciplina_turma      FOREIGN KEY (id_turma) REFERENCES turma(id_turma),
    ADD CONSTRAINT FK_rel_turma_disciplina_disciplina FOREIGN KEY (id_disciplina) REFERENCES disciplina(id_disciplina);
```

```
> db.rel_turma_disciplina.updateOne({_id: ObjectId('60f338614ba1815597a214fc')},{$set:{id_disciplina: ObjectId('5fc54c6eeab2dc17e5e29bd8')}})
```

```
> ALTER TABLE rel_disciplina_curso
    ADD CONSTRAINT FK_rel_disciplina_curso_disciplina FOREIGN KEY (id_disciplina) REFERENCES disciplina(id_disciplina),
    ADD CONSTRAINT FK_rel_disciplina_curso_curso      FOREIGN KEY (id_curso)      REFERENCES curso(id_curso);
```

```
> db.rel_disciplina_curso.updateOne({_id: ObjectId('60f339764ba1815597a214fd')},{$set:{id_curso: ObjectId('5fc54c39eab2dc17e5e29bd6')}})
```

```
> ALTER TABLE rel_avaliacao_turma_disciplina
    ADD CONSTRAINT FK_rel_avaliacao_turma_disciplina_avaliacao       FOREIGN KEY (id_avaliacao)        REFERENCES avaliacao(id_avaliacao),
    ADD CONSTRAINT FK_rel_avaliacao_turma_disciplina_turma_disciplina FOREIGN KEY (id_turma_disciplina) REFERENCES
  rel_turma_disciplina(id_turma_disciplina);
> db.rel_avaliacao_turma_disciplina.updateOne({_id: ObjectId('60f33ee64ba1815597a214ff')}, {$set:{id_turma_disciplina: ObjectId('60f338614ba1815597a214fc')}})
```

```
> ALTER TABLE rel_professor_turma_disciplina
    ADD CONSTRAINT FK_rel_professor_disciplina_professor  FOREIGN KEY (id_usuario)          REFERENCES usuario(id_usuario),
    ADD CONSTRAINT FK_rel_professor_disciplina_disciplina FOREIGN KEY (id_turma_disciplina) REFERENCES rel_turma_disciplina(id_turma_disciplina);
> db.rel_professor_turma_disciplina.updateOne({_id: ObjectId('60f89c8e2ac2d5711ae8896a')},{$set:{id_turma_disciplina: ObjectId('60f338614ba1815597a214fc')}})
```

```
> ALTER TABLE rel_aluno_turma_disciplina
    ADD CONSTRAINT FK_rel_aluno_turma_disciplina_usuario          FOREIGN KEY (id_usuario)          REFERENCES usuario(id_usuario),
    ADD CONSTRAINT FK_rel_aluno_turma_disciplina_turma_disciplina FOREIGN KEY (id_turma_disciplina) REFERENCES
  rel_turma_disciplina(id_turma_disciplina);
> db.rel_aluno_turma_disciplina.updateOne({_id: ObjectId('60f33c394ba1815597a214fe')},
  {$set:{id_turma_disciplina: ObjectId('60f338614ba1815597a214fc'),
    id_usuario: ObjectId('60f314e94ba1815597a214f7')}})
```

```
> ALTER TABLE rel_situacao_aluno_turma
    ADD CONSTRAINT FK_rel_situacao_aluno_disciplina_situacao   FOREIGN KEY (id_situacao)              REFERENCES situacao(id_situacao),
    ADD CONSTRAINT FK_rel_situacao_aluno_disciplina_aluno      FOREIGN KEY (id_usuario)               REFERENCES usuario(id_usuario),
    ADD CONSTRAINT FK_rel_situacao_aluno_turma_disciplina      FOREIGN KEY (id_aluno_turma_disciplina) REFERENCES
  rel_aluno_turma_disciplina(id_aluno_turma_disciplina);
> db.rel_situacao_aluno_turma.updateOne({_id: ObjectId('60f36a3a4ba1815597a21501')},
  {$set:{id_situacao: ObjectId('5fc54cb8eab2dc17e5e29bda'),
  id_usuario: ObjectId('60f314e94ba1815597a214f7'),
  id_aluno_turma_disciplina: ObjectId('60f33c394ba1815597a214fe')
  }})
```

```
> ALTER TABLE rel_avaliacao_turma_aluno
    ADD CONSTRAINT FK_rel_avaliacao_turma_disciplina       FOREIGN KEY (id_avaliacao_turma_disciplina) REFERENCES
  rel_avaliacao_turma_disciplina(id_avaliacao_turma_disciplina),
    ADD CONSTRAINT FK_rel_avaliacao_disciplina_aluno_aluno FOREIGN KEY (id_usuario) REFERENCES usuario(id_usuario);
> db.rel_avaliacao_turma_aluno.updateOne({_id: ObjectId('60f3405d4ba1815597a21500')},
  {$set:{id_usuario: ObjectId('60f314e94ba1815597a214f7'),
  id_avaliacao_turma_disciplina: ObjectId('60f33ee64ba1815597a214ff')
  }})
```

Arquivo JSON

[
{"type":"header","version":"5.0.3","comment":"Export to JSON plugin for PHPMyAdmin"},
{"type":"database","name":"ifspservicos"},
{"type":"table","name":"avaliacao","database":"ifspservicos","data":
[
{"id_avaliacao":"1","data":"2020-11-01","descricao":"Teoria dos Numeros","peso":"5.00"}
]
}
,{"type":"table","name":"curso","database":"ifspservicos","data":
[
{"id_curso":"1","titulo":"Analise e Desenvolvimento de Sistemas","area":"Tecnologia","duracao":"6"}
]
}
,{"type":"table","name":"disciplina","database":"ifspservicos","data":
[
{"id_disciplina":"1","codigo":"AL1","titulo":"Algoritmo 1","carga_horaria":"80"}
]
}

```
,{"type":"table","name":"noticia","database":"ifspservicos","data":
[
{"id_noticia":"1","titulo":"ABC","corpo":"Alfabeto Grego"}
]
}
,{"type":"table","name":"periodo","database":"ifspservicos","data":
[
{"id_periodo":"1","descricao":"Matutino"}
]
}
,{"type":"table","name":"rel_aluno_turma_disciplina","database":"ifspservicos","data":
[
{"id_aluno_turma_disciplina":"1","id_usuario":"1","id_turma_disciplina":"1","frequencia":"70.00"}
]
}
,{"type":"table","name":"rel_avaliacao_turma_aluno","database":"ifspservicos","data":
[
{"id_avaliacao_turma_aluno":"1","id_avaliacao_turma_disciplina":"1","id_usuario":"1","nota":"6.00"}
]
}
,{"type":"table","name":"rel_avaliacao_turma_disciplina","database":"ifspservicos","data":
[
{"id_avaliacao_turma_disciplina":"1","id_avaliacao":"1","id_turma_disciplina":"1"}
]
}
,{"type":"table","name":"rel_disciplina_curso","database":"ifspservicos","data":
[
{"id_disciplina_curso":"1","id_disciplina":"1","id_curso":"1"}
]
}
,{"type":"table","name":"rel_professor_turma_disciplina","database":"ifspservicos","data":
[
{"id_professor_turma_disciplina":"1","id_usuario":"1","id_turma_disciplina":"1"}
]
}
,{"type":"table","name":"rel_situacao_aluno_turma","database":"ifspservicos","data":
[
{"id_situacao_aluno_turma":"1","id_situacao":"1","id_usuario":"1","id_aluno_turma_disciplina":"1","media_final":"10.00"}
]
}
,{"type":"table","name":"rel_turma_disciplina","database":"ifspservicos","data":
[
{"id_turma_disciplina":"1","id_turma":"20151","id_disciplina":"1"}
]
}
,{"type":"table","name":"rel_turma_periodo","database":"ifspservicos","data":
[
{"id_turma_periodo":"1","id_turma":"20151","id_periodo":"1"}
]
}
,{"type":"table","name":"situacao","database":"ifspservicos","data":
[
{"id_situacao":"1","descricao":"Aprovado"}
]
}
,{"type":"table","name":"telefone","database":"ifspservicos","data":
[
{"id_telefone":"1","local":"Hortolandia","ddd":"19","numero":"999999999","area":"Secretaria"}
]
}
,{"type":"table","name":"tipo_usuario","database":"ifspservicos","data":
[
```

{"id_tipo_usuario":"1","descricao":"Aluno"}
]
}
,{"type":"table","name":"turma","database":"ifspservicos","data":
[
{"id_turma":"20151","ano":"2015","semestre":"1"}
]
}
,{"type":"table","name":"usuario","database":"ifspservicos","data":
[
{"id_usuario":"1","id_tipo_usuario":"1","nome":"Dafne","rg":"999999999","cpf":"99999999999","cep":"99999999","telefone
":"9999999999","celular":"99999999999","email":"dafne@aluno.ifsp.com","ra":"999999","senha":"ifsp@123","token":"ifsp
@123"}
]
}
]