UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

THOMAS ANDERSON FEITOSA MONTEIRO

MODELAGEM DE BANCO DE DADOS ORIENTADOS A DOCUMENTOS COM AML

RECIFE

2023

UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

THOMAS ANDERSON FEITOSA MONTEIRO

MODELAGEM DE BANCO DE DADOS ORIENTADOS A DOCUMENTOS COM AML

Monografia apresentada ao Centro de Informática (CIn) da Universidade Federal de Pernambuco (UFPE), como requisito parcial para conclusão do Curso de Ciência da Computação, orientada pelo professor Robson do Nascimento Fidalgo.

RECIFE
2023

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Monteiro, Thomas Anderson Feitosa.
Modelagem de banco de dados orientados a documentos com AML / Thomas Anderson Feitosa Monteiro. - Recife, 2023.
40 p : il.

Orientador(a): Robson do Nascimento Fidalgo
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Ciências da Computação - Bacharelado, 2023.

1. AML. 2. Modelagem lógica. 3. NoSQL. 4. DDD. 5. Agregados. I. Fidalgo, Robson do Nascimento. (Orientação). II. Título.

000 CDD (22.ed.)

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

THOMAS ANDERSON FEITOSA MONTEIRO

MODELAGEM DE BANCO DE DADOS ORIENTADOS A DOCUMENTOS COM AML

Monografia submetida ao corpo docente da Universidade Federal de Pernambuco, defendida e aprovada em 27 de setembro de 2023.

Banca Examinadora:

Orientador(a)

Robson do Nascimento Fidalgo

Doutor(a)

Examinador(a)

Vinicius Cardoso Garcia

Doutor(a)

AGRADECIMENTOS

Entrego este trabalho de conclusão representando um coletivo de pessoas que me deram suporte, devido principalmente ao fato de nenhuma delas ter dado a opção de desistir do curso ao longo das dificuldades. A própria matrícula não seria possível sem o apoio dos meus tios que forneceram moradia durante anos até me tornar independente. Tive grande apoio da família, de amigos que conheci durante o curso que foram fundamentais, de colegas de trabalho e também dos excelentes professores do CIN (Centro de Informática). Chego à reta final contando com o apoio do Prof. Robson que me orienta e seu doutorando Genesis sempre tirando dúvidas sobre o tema do trabalho. Um enorme conjunto de pessoas tornaram possível essa caminhada e só tenho a agradecer a todos.

“Odeio o privilégio e o monopólio. Para mim, tudo o que não pode ser dividido com as multidões é tabu.”
Mahatma Gandhi

RESUMO

Nos últimos anos houve, devido a ascensão de grandes redes sociais, big data entre outros uma ascensão de bancos NoSQL. Dentre estes bancos definidos como NoSQL destacam-se os bancos orientados a documentos, em especial o MongoDB com grande popularização. À medida que mais organizações adotam bancos orientados a documentos, há uma necessidade crescente de diretrizes e melhores práticas na modelagem lógica desses sistemas. Uma forma de criar modelagem de dados lógica para este tipo de banco é através da Aggregate Modeling Language (AML), que tem forte base no conceito de agregados do Domain-Driven Design.

Palavras-chave: NoSQL, Modelagem lógica, DDD, AML, Agregados

ABSTRACT

In recent years, due to the rise of large social networks, big data, and other factors, there has been an emergence of NoSQL databases. Among these databases classified as NoSQL, document-oriented databases have gained prominence, with MongoDB being particularly popular. As more organizations adopt document-oriented databases, there is a growing need for guidelines and best practices in the logical modeling of these systems. One approach to creating logical data modeling for this type of database is through the use of the Aggregate Modeling Language (AML), which is strongly rooted in the concept of aggregates from Domain-Driven Design.

Keywords: NoSQL, Logical modeling, DDD, AML, Aggregates

Sumário

1. Introdução..... 7
1.1. Motivação.....7
1.2. Objetivos..... 7
2. Conceitos Básicos.....8
2.1. Importância da modelagem de dados.....8
2.2. Bancos NoSQL.....11
2.3. Bancos orientados a documentos.....12
2.4. AML - Aggregate Modelling Language..... 14
2.4.1. Construtores da AML..... 17
Atributos.....17
Entidades e objetos de valor.....19
Agregados.....20
Links.....21
3. Boas práticas para modelagem de bancos orientados a documentos..... 25
3.1. Modelagem de agregados com AML..... 25
4 Conclusão.....38

Tabela de Siglas

Sigla	Significado
SQL	Structured Query Language
UML	Unified Modeling Language
DDD	Domain Driven Design
ER	Diagrama entidade-relacionamento
OMG	Object Management Group
DSML	Domain-Specific Modelling Language

1. Introdução

1.1. Motivação

Os bancos de dados orientados a documentos estão se tornando cada vez mais prevalentes em uma ampla variedade de aplicações e sistemas, devido à sua flexibilidade e escalabilidade. Isso torna a pesquisa e o desenvolvimento nessa área altamente relevantes para o cenário atual da computação.

A modelagem de bancos de dados orientados a documentos apresenta desafios distintos em comparação com os sistemas tradicionais de banco de dados relacional. Explorar esses desafios proporciona uma oportunidade valiosa para entender e resolver questões específicas que surgem nesse contexto.

À medida que mais organizações adotam bancos orientados a documentos, há uma necessidade crescente de diretrizes e melhores práticas na modelagem lógica desses sistemas. O presente trabalho visa contribuir para a definição dessas práticas e padrões.

1.2. Objetivos

O objetivo deste trabalho é mostrar uma abordagem prática de modelagem lógica de bancos orientados a documentos utilizando Aggregate Modeling Language (AML). Será mostrado como utilizar construtores AML no processo de modelagem lógica a partir do conceito de modelagem orientada a agregados, um conceito do DDD que pode ser utilizado para balizar a modelagem de dados. O trabalho também irá mostrar, a partir de um exemplo de uso do mundo real, cenários em que uma abordagem de modelagem se torna mais eficaz frente a outra do ponto de vista de desempenho e consistência. Este trabalho se limita apenas ao contexto de bancos orientados a documentos, utilizando como referência para exemplos o banco MongoDB devido ao fato de ser o banco orientado a documentos mais utilizado no momento em que este trabalho é desenvolvido.

2. Conceitos Básicos

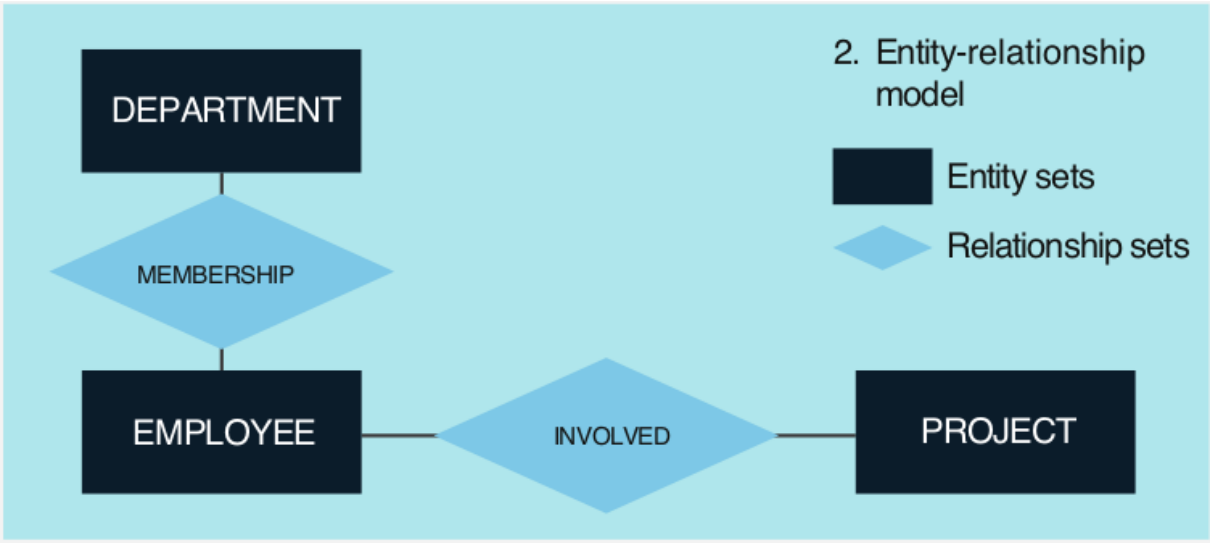
Neste capítulo, são introduzidos alguns termos e conceitos utilizados ao longo deste trabalho, em especial o por que engenheiros de software, administradores de bancos de dados e arquitetos de sistemas buscam meios de modelagem como ferramenta de auxílio à implementação do trabalho. Também são discutidas as diferenças e desafios específicos de modelagem de bancos não relacionais quando comparados a bancos relacionais.

2.1. Importância da modelagem de dados

Na área de bancos de dados a modelagem de dados tem um papel fundamental, pois a partir dessa é possível tirar conclusões a respeito de como estruturar o banco. Michael Kaufmann e Andreas Meier [2] em sua publicação afirmam que o processo de estruturação de um banco de dados se constitui em 3 fases principais: análise de requisitos, modelagem conceitual e a implementação dos esquemas do banco. A modelagem conceitual tem um papel importante principalmente em se tratando de documentação de conhecimento.

Quando se fala de modelagem de bancos de dados existem 3 tipos distintos: modelagem conceitual, lógica e física. O modelo conceitual é o de mais alto nível e por ser menos detalhado possibilita uma comunicação e discussão efetiva com pessoas de negócio e menos técnicas antes das etapas de implementação. Nessa modelagem inicial geralmente são definidas as entidades e relacionamentos entre as mesmas.

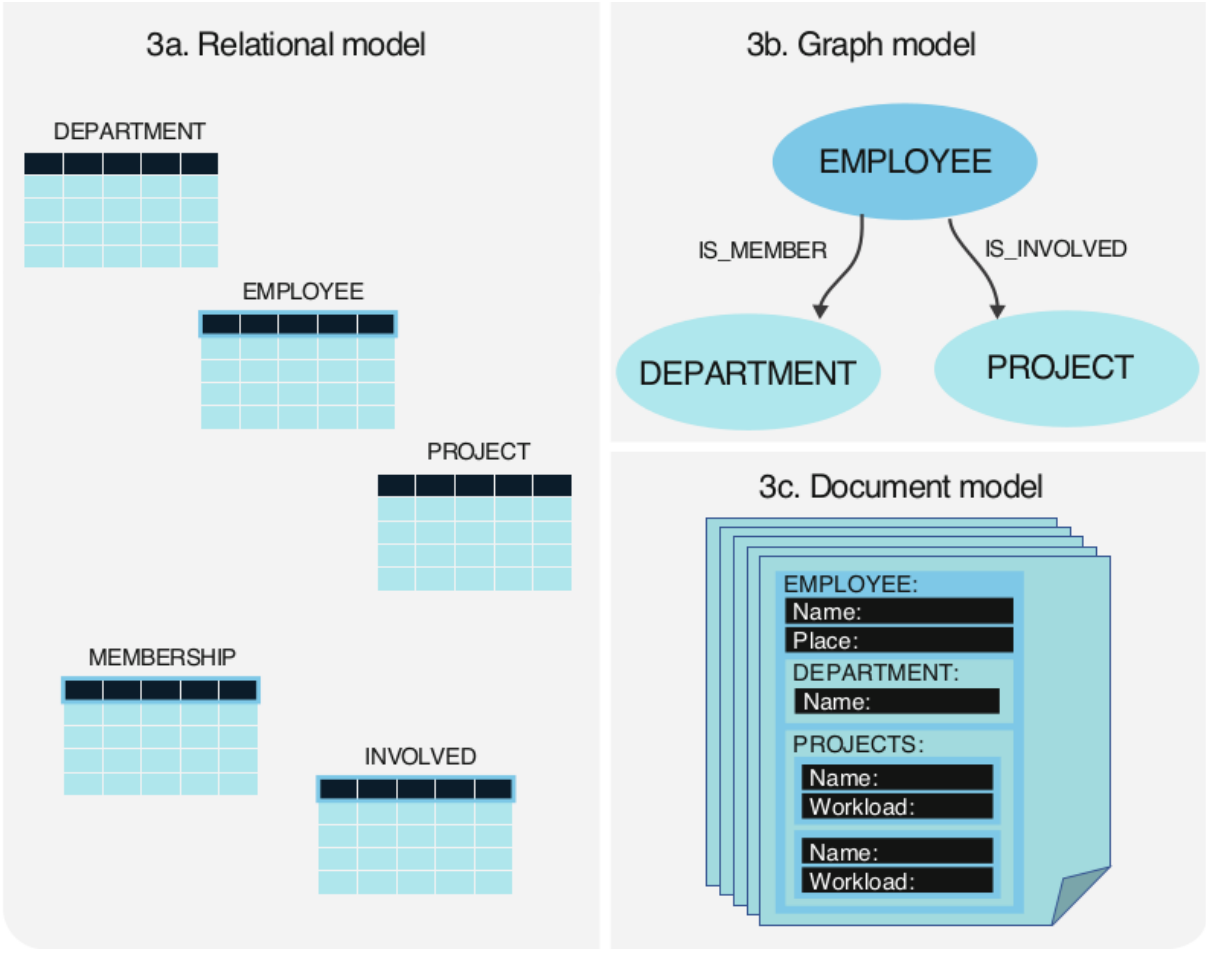
Figura 1 - Modelagem conceitual envolvendo 3 entidades



Fonte: Figura de [2]

A modelagem lógica por outro lado contém detalhes estruturais do banco utilizado. O mapeamento deste modelo conceitual apresentado em um modelo lógico deve ser feito em uma notação adequada a depender do paradigma do banco utilizado.

Figura 2 - Modelagens lógicas em 3 paradigmas de bancos distintos



Fonte: Figura de [2]

Observando a Figura 2, no banco de dados relacional, os relacionamentos, como esperado por serem N para N, aparecem como tabelas, o que não existe por exemplo no banco de dados de grafos, pelas referências de relacionamento existirem nos próprios nós (ou documentos). Particularmente interessante é exemplo em 3c que mostra o modelo lógico para um banco de dados não relacional orientado a documentos, pois nada no modelo conceitual indica que essa seria a hierarquia dos objetos no modelo lógico. Qualquer das entidades poderiam estar hierarquicamente no nível mais alto e representar os mesmos relacionamentos. Michael Kaufmann e Andreas Meier [2] deixam claro que essa hierarquia na modelagem 3c foi uma escolha particular, não algo que derivou da etapa de modelagem conceitual.

Esta variedade de possibilidades na representação de entidades em um banco orientado a documentos torna especialmente importante além da modelagem conceitual também a modelagem lógica que é o foco deste trabalho.

2.2. Bancos NoSQL

Por um tempo os bancos relacionais, eram a escolha indiscutível em se tratando de bancos de dados. Até os dias atuais os bancos de dados relacionais são largamente utilizados e tem soluções bastante maduras e testadas no mercado como Oracle, Postgres, SQL Server entre outros. De acordo com o site statista.com [4] até fevereiro deste ano (2023) dos 5 bancos de dados mais utilizados, os 4 primeiros são bancos de dados relacionais, o que mostra sua importância no mercado.

Embora tecnicamente o modelo relacional tenha muito a oferecer, com a popularização cada vez maior da internet e grandes redes sociais - com sua grande quantidade de dados gerados - acabaram por demandar bancos mais flexíveis e eficientes para novos cenários. Dentre estes cenários temos como por exemplo análise de dados, armazenamento de grandes quantidades de logs ou tempos de leitura próximo ao real time. Cada uma dessas necessidades acabou por resultar em bancos com abordagens distintas com foco na solução de problemas específicos, com muitos destes sendo categorizados como bancos NoSQL.

Sadalage e Fowler [6] apontam que não existe uma definição geral aceita para o termo NoSQL, porém listam uma série de características para categorizar este tipo de banco.

> *"Não existe uma definição geral aceita, nem uma entidade para fornecer uma, então tudo o que podemos fazer é discutir algumas características comuns dos bancos de dados que tendem a ser chamados de NoSQL." [13, tradução própria]*

Dentre essas características destacam-se a ausência da linguagem SQL e o fato de operar sem necessidade de definição de um esquema prévio dos dados a serem inseridos.

2.3. Bancos orientados a documentos

Os bancos orientados a documentos são um tipo específico de banco NoSQL. Sadalage e Fowler [6] descrevem os documentos suportados por este tipo de banco como estruturas hierárquicas em árvore, sendo estas estruturas compostas por mapas chave-valor, coleções e escalares. Uma dessas estruturas hierárquicas largamente conhecidas é o JSON.

Figura 3 - Representação de documento JSON

```
// coleção::Usuário

{
  "_id": 1,
  "nome": "Pedro",
  "permissões": ["EXCLUSÃO", "VISUALIZAÇÃO", "EDIÇÃO"],
  "endereço": {
    "rua": "Av. Recife",
    "número": "100"
  }
}
```

Fonte: De autoria própria

Em termos de popularidade, o banco de dados orientado a documentos mais utilizado atualmente [4] é o MongoDB.

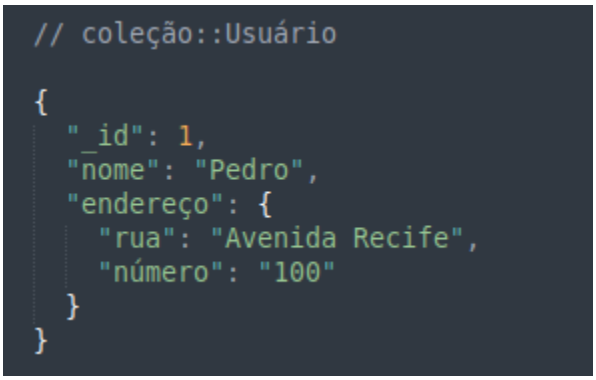
Bancos orientados a documentos têm por característica serem *schemaless* [6], ou seja, oferecem suporte ao armazenamento e consulta de dados sem restrição estrutural em uma mesma coleção. Essa permissividade de dados com estruturas distintas traz consigo decisões a serem tomadas na modelagem lógica dos dados no banco. Alessandro Fiori [3] diz que por conta dessas particularidades, especialmente a heterogeneidade dos dados, a modelagem se torna mais uma arte que um mero exercício de engenharia, isto porque o design final da modelagem lógica deve levar em conta casos de uso da aplicação.

Uma particularidade na modelagem de bancos orientados a documentos diz respeito a como relacionar seus dados. Basicamente existem duas formas [9]: pela utilização de documentos embutidos ou documentos referenciados. Nos relacionamentos implementados via documentos embutidos, a relação ocorre por hierarquia na árvore de objetos em uma

mesma coleção de dados. Por sua vez, no relacionamento por referência, os documentos residem em coleções distintas com referências ao documento relacionado.

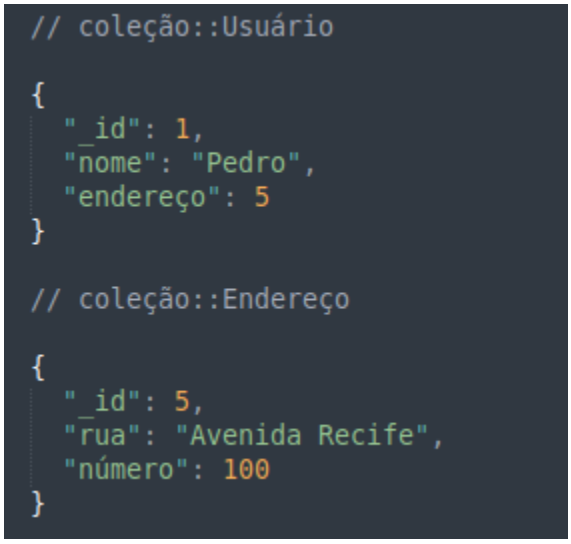
Exemplo de duas possibilidades distintas de relação 1 para 1 em um banco orientado a documentos:

Figura 4 - Representação de documento JSON com relacionamento embutido de entidades Usuário e Endereço



Fonte: De autoria própria

Figura 5 - Representação de documento JSON com relacionamento por referência de entidades Usuário e Endereço



Fonte: De autoria própria

2.4. AML - Aggregate Modelling Language

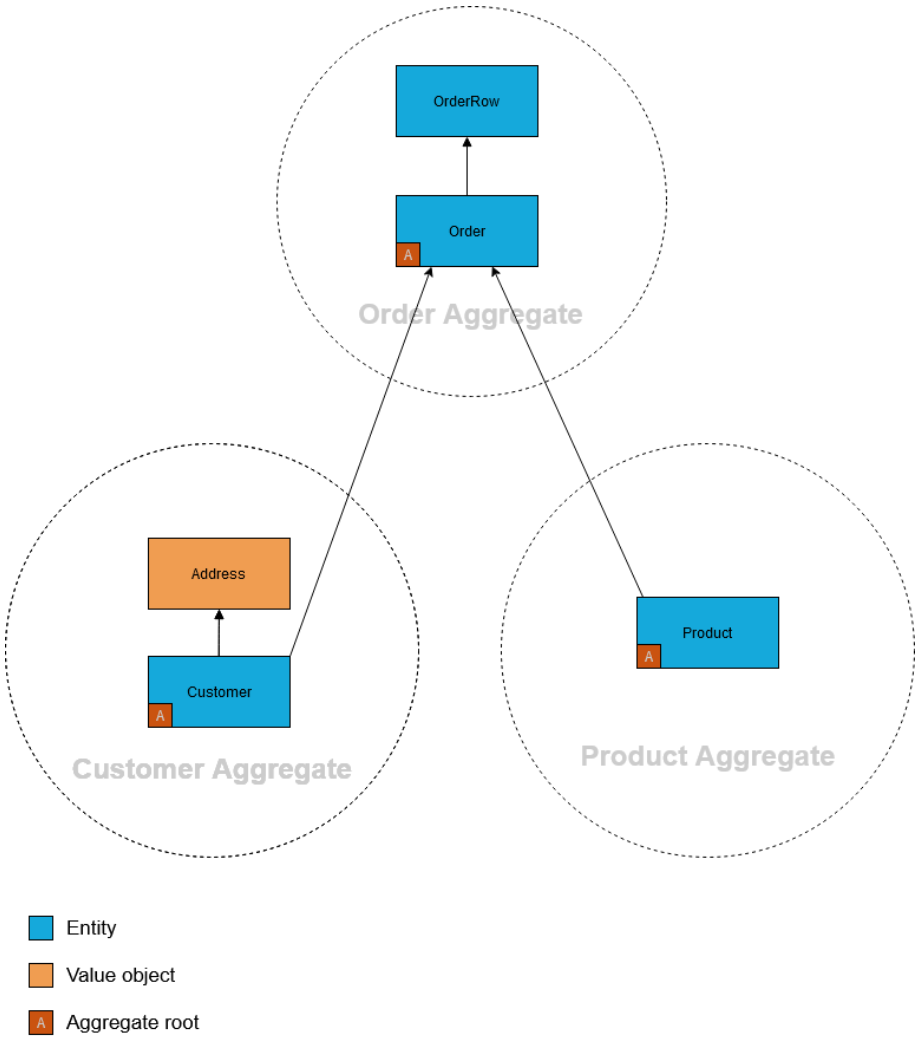
A necessidade de lidar com hierarquias quando modelando a estrutura de um banco orientado a documentos traz elementos a mais de dificuldade. Uma dessas dificuldades é a tomada de decisão em termos de estrutura, ou seja, quando dois documentos distintos devem estar aninhados na mesma coleção e quando devem ser persistidos em coleções diferentes. Hoberman [12] sugere, por exemplo, um conjunto de heurísticas a se considerar na tomada dessas decisões de modelagem, como a dependência existencial entre entidades e frequência com que são lidas simultaneamente. Por ser um tema pouco explorado [11] quando se trata de modelagem lógica para bancos NoSQL, outro tema igualmente importante é a representação diagramática dessas hierarquias em uma linguagem de modelagem.

Para o primeiro desafio, um ponto de partida que alguns autores como Sadalage e Fowler [6] citam para tomada dessas decisões é o DDD (Domain-Driven Design), especialmente seu conceito de entidades e agregados. Segundo esses autores, o conceito de agregados pode guiar o processo de modelagem lógica de bancos orientados a documentos, especialmente como método de modelagem com fins de reduzir ou evitar a necessidade de joins.

No DDD - que é utilizado como guia de modelagem de desenvolvimento orientado a objetos - uma entidade é uma classe que representa objetos que tenham identidade própria. Vaughn Vernon [5] em seu livro *Implementando DDD* define uma entidade nos seguintes termos: é a identidade única e a mutabilidade de características que distinguem entidade de objetos de valor. Um exemplo disso seria uma classe representando um usuário, que tem identidade única representando uma pessoa, diferente disso seria uma classe para representar e agrupar informações do endereço do usuário, por não possuir identidade única é considerado um objeto de valor.

Além desta distinção entre entidades e objetos de valor, o DDD também define o conceito de agregados. Vernon [5] descreve os agregados como clusters de objetos interligados entre si. Evans [1] cita que um agregado pode ter apenas uma entidade como raiz, a qual preferencialmente deve servir como ponto inicial de acesso a qualquer objeto dentro daquele agregado.

Figura 6 - Exemplo de 3 agregados no DDD com suas entidades e objetos de valor



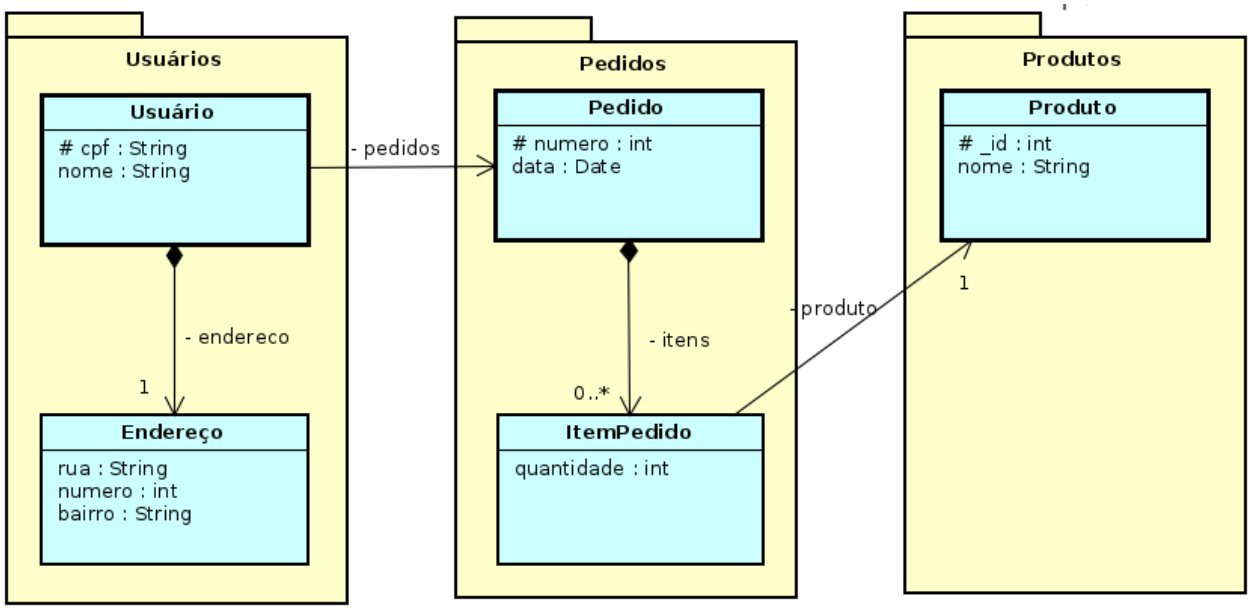
Fonte: Retirado de [10]

Sadalage e Fowler [6] em NoSQL distilled discutem o fato de que alguns bancos NoSQL, como por exemplo os bancos orientados a documentos, possuírem a característica de serem "orientados a agregados". Essa característica identificada se dá pelo fato de nesse tipo de banco existir uma tendência de que os dados relacionados se mantenham agrupados na mesma coleção, uma ligação semelhante ao que ocorre com os agregados do DDD. Segundo Sadalage e Fowler ter essa abordagem de modelagem do banco orientada a agregados, tomando como base os mesmos agregados do DDD, pode trazer consigo alguns benefícios. Modelar um agregado em uma mesma coleção do banco pode levar a uma maior facilidade ao lidar com clusters de instâncias diversas do banco, diminuindo o número de nós distintos necessários na busca de informações. Outro benefício citado é a obtenção de atomicidade em operações de leitura e escrita das informações desses agregados, levando a ganhos no quesito de consistência de dados.

No contexto de modelagem de dados, uma das formas de representação lógica destes agregados se dá a partir da representação estrutural dos documentos como na Figura 2. Este tipo de representação apesar de ilustrar bem a estrutura deste agregado e as hierarquias no documento não fornece uma boa representação de relacionamentos por referência assim como da multiplicidade de relacionamentos por documentos embutidos. Este tipo de modelo é amplamente utilizado em materiais sobre modelagem lógica de bancos orientados a documentos.

Uma linguagem de modelagem que fornece suporte para modelagem de bancos orientados a documentos e que é fortemente baseada no conceito de agregados é a AML. A AML é uma Domain Specific Modeling Language (DSML) inspirada pelos mesmos elementos visuais da UML porém voltada à modelagem de bancos orientados a documentos, o que permite a utilização de ferramentas de modelagem UML já existentes no mercado (e.g, Astah UML).

Figura 7: Exemplo de modelo lógico em AML com 3 coleções



Fonte: De autoria própria

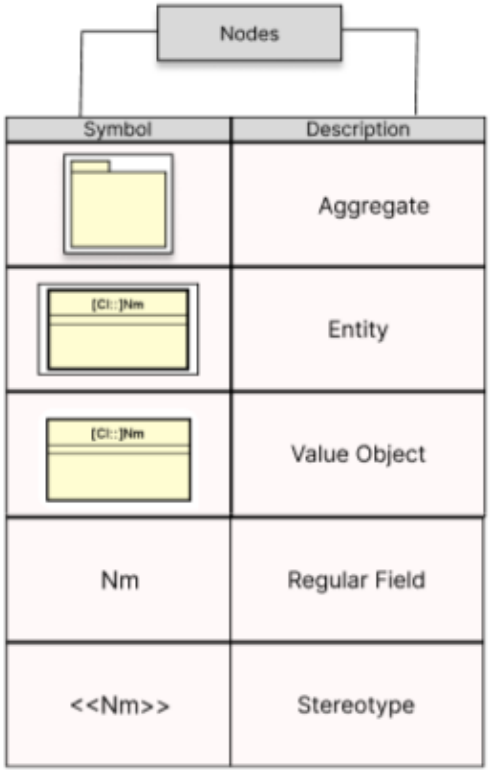
Linguagens de modelagem de propósito geral tem como principal objetivo permitir especificar uma miríade de cenários utilizando seu arcabouço de construtores, já que não orienta-se por um paradigma específico. Diferente da UML, a AML possui uma quantidade

menor de construtores, possui semântica única para cada construtor e menor complexidade. A base sólida no conceito de agregados do Domain-Driven Design (DDD) proporciona um alicerce conceitual robusto, promovendo a coerência e a clareza na modelagem lógica de dados no contexto de bancos orientados a documentos.

2.4.1. Construtores da AML

Os construtores da AML são classificados em nós, links e pictogramas. Por intermédio dos Nós representam-se os conceitos de Atributo (Attribute), Entidade (Entity), VO (Value-Object) , Agregado (Aggregate), e Rótulo Final (Label Final).

Figura 8 - Significado dos elementos gráficos de nós da UML na AML

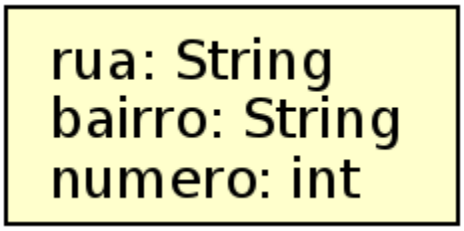


Fonte: Arquivo do idealizador da AML

Atributos

A notação de atributo descreve um campo de uma entidade ou objeto de valor, contendo em sua estrutura mais básica o nome deste atributo seguido do tipo do mesmo.

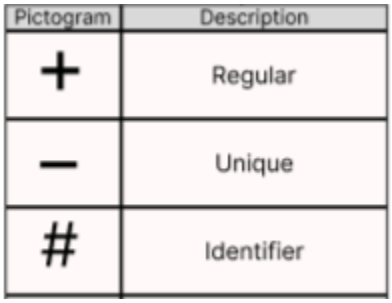
Figura 9 - Declaração de 3 atributos distintos em sua forma mínima, isto é, contendo apenas o nome do campo e o tipo



Fonte: De autoria própria

Além do nome e tipo de campo mostrados na Figura 9, a AML permite, assim como na UML, a definição de multiplicidade de um atributo indicando, no contexto da AML, se o campo em questão se trata de um array. Também é possível a utilização do modificador de visibilidade de atributo da UML, com significados distintos na AML, indicados na Figura 10.

Figura 10 - Significado de elementos de visibilidade de atributo da UML na AML

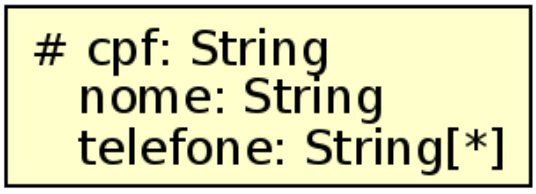


Fonte: Arquivo do idealizador da AML

Na Figura 10, temos o pictograma de visibilidade privada na UML com significado na AML de atributo Unique, isto é, não poder conter referências repetidas para um mesmo objeto. O pictograma Identifier na mesma figura, indica que este campo é o identificador da entidade, ou seja, um atributo que tem valor único entre todas as instâncias da entidade e que, por ter essa propriedade, é definido como identificador da mesma. Vale destacar, que, diferente dos atributos únicos, os atributos identificadores são usados para referenciar documentos. O pictograma Regular por outro lado indica a ausência destas duas restrições e é

opcional, considera-se que um atributo que não utilize o pictograma Unique ou Identifier é um atributo Regular por padrão.

Figura 11 - Exemplo de atributo telefone como array de String e campo cpf declarado como identificador



Fonte: De autoria própria

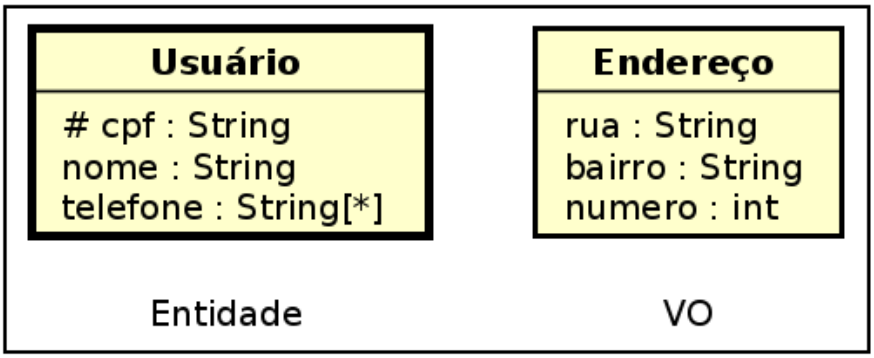
A definição de multiplicidade do atributo telefone na Figura 11 implica que este campo é um array do tipo especificado, no caso, um array de String possuindo N telefones. O uso de Unique no campo cpf indica que este campo é o identificador único entre todas as instâncias de objetos deste mesmo tipo.

O uso do pictograma Unique será exemplificado em conjunto com a utilização de links.

Entidades e objetos de valor

No DDD [1], como discutido na Seção 2, uma entidade é uma classe que representa objetos que tenham identidade própria. Sua representação gráfica na AML se dá como uma figura de classe ativa da UML, isto é, possui uma borda mais grossa. Os Value Objects (VO) , por outro lado, não contém identificador único e na AML são representados como uma classe comum, com a borda mais fina em relação às entidades.

Figura 12 - Exemplos de uma entidade e um Value Object (VO)



Fonte: De autoria própria

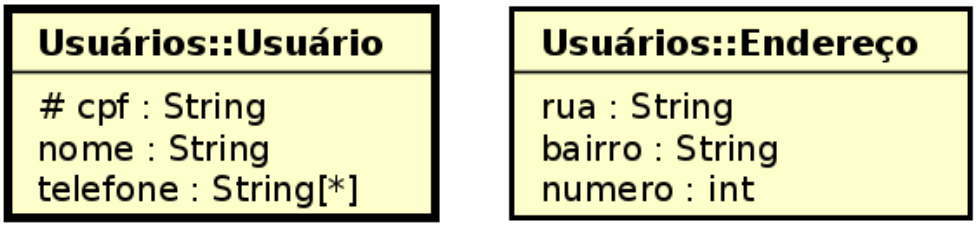
Como mostra a Figura 12, tanto entidades, como Value Objects (VO) são compostos por um Nome (Nm na Figura 8) e seus respectivos atributos, a diferença entre ambos se dá pela identidade única presente nas entidades e que não está presente nos Value Objects. Na Figura 12, usuário é uma entidade e possui o atributo cpf como identificador.

Agregados

Na AML, o uso de agregados é fundamental para modelagem lógica de bancos orientados a documentos, isto porque, na AML existe uma relação direta entre uma coleção em termos de banco de dados e um agregado. O fato de entidades e objetos de valor distintos estarem no mesmo agregado no modelo em AML, implica que a nível de banco de dados estarão presentes em uma mesma coleção.

Existem duas formas de representar agregados na AML. A primeira como prefixo do nome das entidades e Value Objects, seguido pelos caracteres :: que na UML é usado para representar o pacote que a classe está contida.

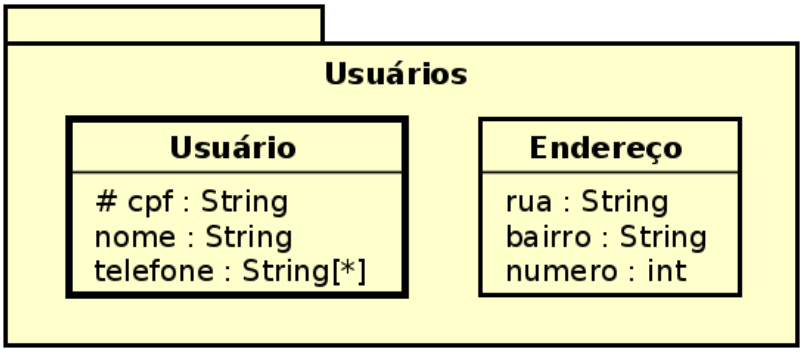
Figura 13 - Exemplo de agregado composto por entidade Usuário e Value Object (VO) Endereço



Fonte: De autoria própria

A segunda forma de representar um agregado em AML é através do símbolo Aggregate visto na Figura 8. Nesta forma, o nome do agregado é sinalizado pelo nome definido no símbolo de Aggregate, sendo desnecessário explicitar o nome do agregado nos nomes das Entidades ou VOs como na Figura 13. Para os exemplos daqui em diante será utilizada esta segunda forma de representação de agregados.

Figura 14 - Exemplo de agregado utilizando símbolo Aggregate



Fonte: De autoria própria

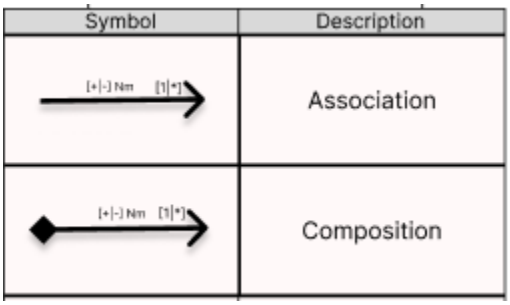
O modelo da Figura 14 é equivalente à Figura 13, ambos contém um único agregado de nome Usuários composto pela mesma entidade Usuário e o mesmo VO Endereço.

Embora a partir da Figura 14 e da Figura 13 seja possível definir a coleção no banco e até mesmo os objetos presentes nesta coleção, estas figuras não definem a hierarquia destes objetos, isto é, como eles se relacionam. Os relacionamentos entre objetos e hierarquia, no caso de objetos de um mesmo agregado, são definidos a partir do uso dos Links.

Links

Os links na AML definem os relacionamentos entre entidades e objetos de valor e existem tipos de links distintos para diferentes situações.

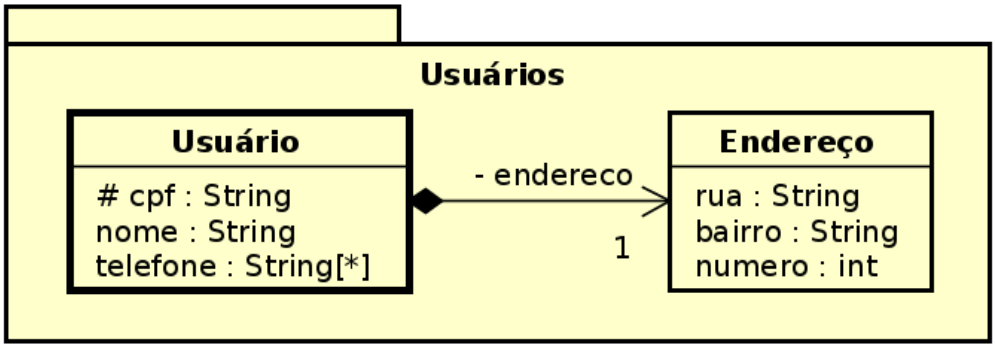
Figura 15 - Links na AML



Fonte: Arquivo do idealizador da AML

Tomando como exemplo a Figura 14, a forma como relacionamos entidades e Value Objects é através do link de composição.

Figura 16 - Link de composição relacionando Entidade e Value Object

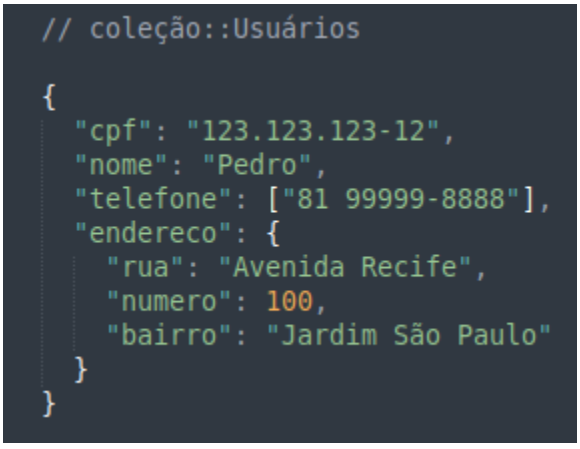


Fonte: De autoria própria

Na Figura 16, temos um exemplo completo de um modelo lógico em AML, com todas as informações necessárias. Temos neste exemplo informações a respeito do nome da coleção, os nomes e atributos da entidade e do VO bem como informações sobre como se relacionam neste agregado (hierarquia, representada pela direção da associação e cardinalidade).

O link de composição define um relacionamento por documento embutido, ele é usado entre Entidade e Value Object e sempre dentro de um mesmo agregado, ou seja, relaciona objetos dentro de uma mesma coleção. A Figura 17 mostra um objeto JSON válido para o modelo da Figura 16 com a relação entre Usuário e Endereço através de documentos embutidos.

Figura 17 - Link de composição relacionando Entidade e Value Object

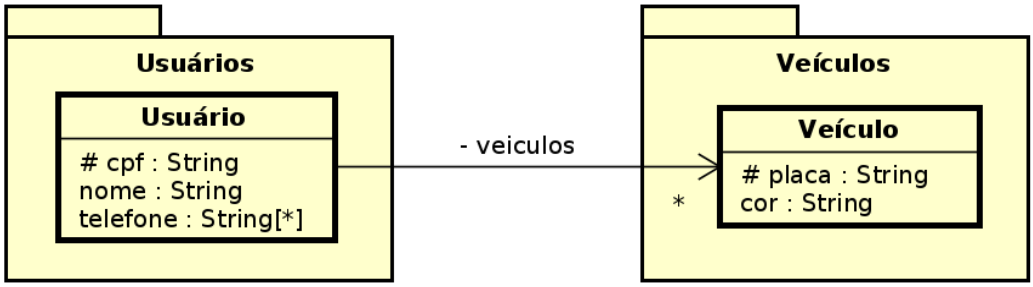


Fonte: De autoria própria

Neste exemplo da Figura 16, Usuário é a raíz do agregado Usuários, por ser o elemento hierarquicamente superior, ou seja, não existe nenhum link partindo em direção a esta entidade.

Outra forma de relacionar objetos na AML é através do uso de referências para objetos que compõem agregados distintos.

Figura 18 - Relacionamento por associação entre entidades de agregados distintos



Fonte: De autoria própria

Diferente da Figura 17, o relacionamento por referência mostrado na Figura 18 é feito com uso do link de associação na AML. Os documentos neste tipo de relacionamento guardam informação apenas do identificador do objeto que referenciam, como mostra a Figura 19.

Figura 19 - JSON exemplificando relacionamento por referência entre Usuário e Veículo

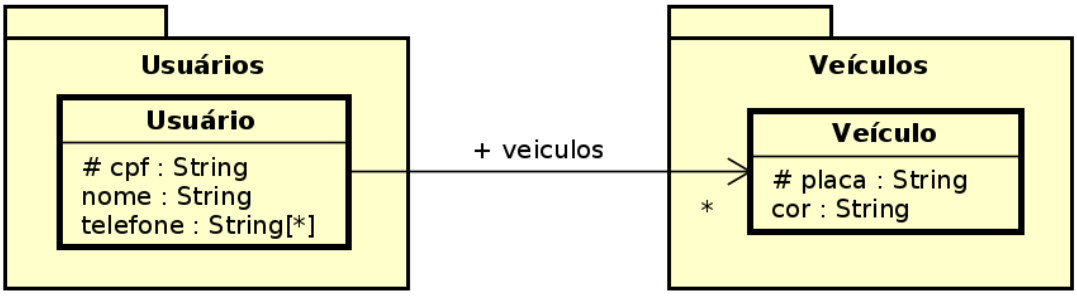
```
// coleção::Usuários

{
  "cpf": "123.123.123-12",
  "nome": "Pedro",
  "telefone": ["81 99999-8888"],
  "veículos": ["ABC-1234"]
}

// coleção::Veículos

{
  "placa": "ABC-1234",
  "cor": "Azul"
}
```

Fonte: De autoria própria

Para permitir instâncias repetidas de um mesmo objeto em um relacionamento deve-se substituir o pictograma Unique no link que relaciona as entidades pelo pictograma Regular da Figura 10. Isto é, na Figura 18 tem-se a representação gráfica de um relacionamento 1:N pois "veiculos" é precedido pelo pictograma Unique (-veiculos), enquanto que na Figura 20 tem-se

a diagramação de um relacionamento M:N, pois não há essa restrição de unicidade, uma vez que utilizado o Pictograma Regular (+veiculos).

Figura 20 - Relacionamento por referência sem restrição de instâncias duplicadas



Fonte: De autoria própria

3. Boas práticas para modelagem de bancos orientados a documentos

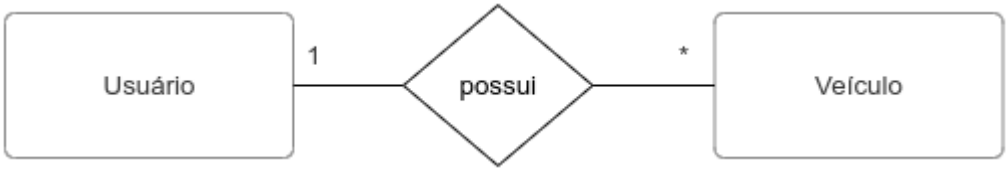
Este capítulo tem como objetivo introduzir o leitor ao processo de modelagem lógica de dados de bancos orientados a documentos a partir do uso da Aggregate Modeling Language (AML) e seus construtores, bem como auxiliar na aplicação de boas práticas de modelagem. Serão apresentadas variações distintas de modelagem de relacionamentos, contribuindo para a identificação de *bad smells* relativos a modelagem de dados orientados a documentos a partir da reflexão sobre as vantagens e desvantagens entre diferentes modelagens lógicas de um determinado domínio. Serão levantadas as implicações dessas escolhas na construção e evolução de uma aplicação em termos de desempenho e consistência. Os exemplos utilizam a linguagem de modelagem lógica AML e para representação dos documentos o formato JSON utilizado no MongoDB.

3.1. Modelagem de agregados com AML

A AML permite variações distintas de modelagens lógicas para um mesmo esquema conceitual. Este capítulo tem como objetivo mostrar a modelagem de relacionamentos utilizando AML, tanto por meio de documentos embutidos quanto fazendo uso de documentos referenciados, discutindo quando utilizar cada modelo possível.

Tomemos como exemplo esquema conceitual a seguir, onde um usuário pode possuir vários veículos e um veículo pode ser possuído por no máximo um usuário.

Figura 21 - Modelo conceitual utilizado como base para modelagem lógica

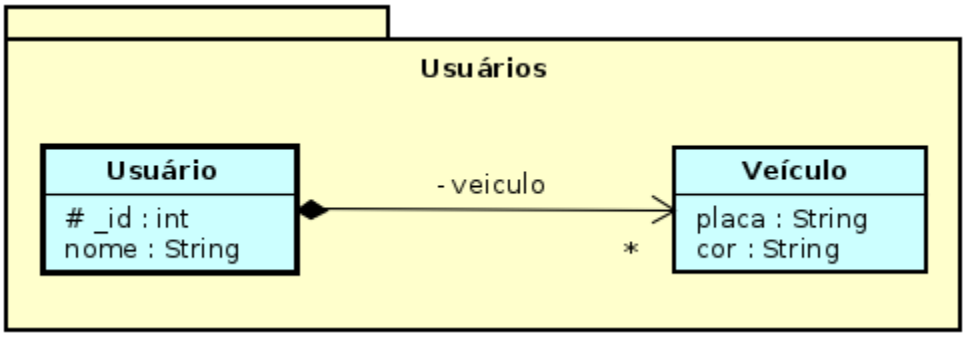


Fonte: De autoria própria

Para este modelo conceitual existem as 8 seguintes possibilidades de modelagem lógicas em um banco orientado a documentos:

1. **Coleção com array de documentos embutidos**

Figura 22- Coleção de usuários contendo N veículos



Fonte: De autoria própria

Na Figura 22 existe apenas um agregado em que Usuário é a raiz do mesmo. Logo, o ciclo de vida dos documentos representando os veículos estão diretamente relacionados à existência dos objetos de usuários. Este relacionamento utilizando documentos embutidos garante que os documentos serão armazenados fisicamente juntos a nível de disco e que podem ser encontrados a partir da mesma instância do banco [9], contribuindo no desempenho de leitura e escrita especialmente em casos de buscas a partir dos usuários. Além disso, esse relacionamento, por atrelar a existência dos documentos de veículos ao usuário ao qual pertence, garante a nível de banco uma das características dos agregados no DDD [1], que é restringir (quando possível) o acesso às entidades somente a partir da raiz dos agregados da mesma.

Em forma de JSON este tipo de agregado consiste de uma única coleção:

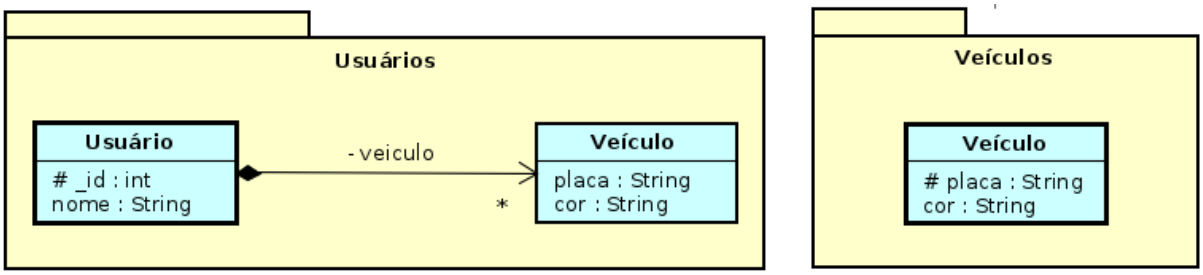
Figura 23 - Representação em JSON de coleção de usuários contento N veículos

```
// coleção::Usuário

{
  "_id": 1,
  "nome": "John",
  "veículos": {
    "placa": "ABC1234",
    "cor": "preto"
  }
}
```

Fonte: De autoria própria

2. Coleção com array de documentos embutidos mais coleção redundante

Figura 24 - Coleção de usuários contento N veículos com redundância dos documentos de veículos



Fonte: De autoria própria

Na Figura 24 existem 2 agregados independentes, o agregado à esquerda tendo a entidade Usuário como raíz e o segundo tendo a entidade Veículo como raíz e única entidade do agregado. No primeiro agregado a relação entre as entidades é de composição, pelo fato de veículo estar estruturalmente atrelado à existência das instâncias dos usuários. A imagem a seguir mostra a estrutura JSON dessas coleções:

Figura 25 - Estrutura JSON de modelagem com 2 agregados

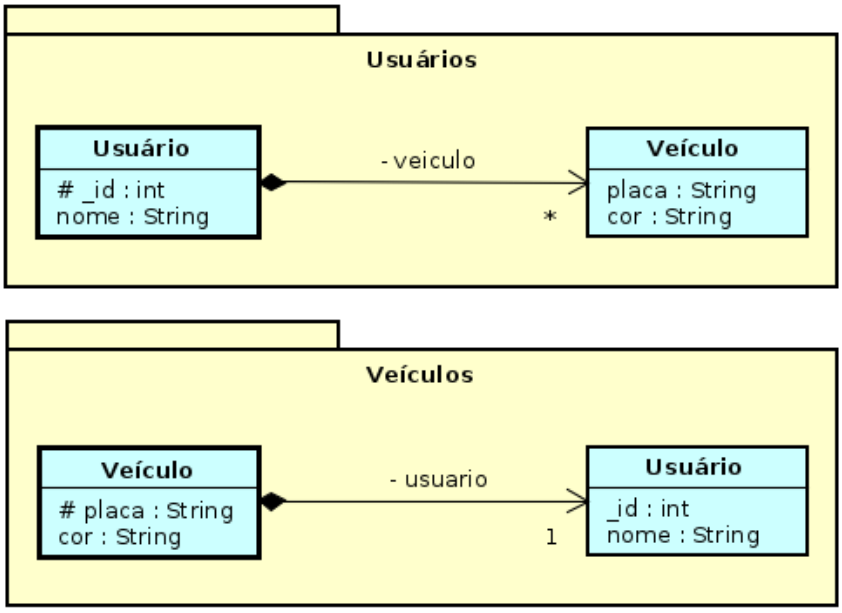
```
// coleção::Usuário

{
  "_id": 1,
  "nome": "John",
  "veículos": {
    "placa": "ABC1234",
    "cor": "preto"
  }
}

// coleção::Veículo

{
  "placa": "ABC1234",
  "cor": "preto "
}
```

Fonte: De autoria própria

Esta modelagem pode ser utilizada em um cenário que a existência da entidade veículo seja independente à existência de algum usuário específico, ou seja, independe do relacionamento no primeiro agregado. Essa redundância de dados pode levar a problemas de inconsistência ao lidar com atualizações e exclusões de documentos - que era uma das vantagens do primeiro modelo - porém têm como benefício ganhos de desempenho. Rick Copeland [9] fala sobre esses ganhos de desempenho obtidos através de redundância de dados no MongoDB, porém deixando claro que essa redundância pode dificultar o processo de design do modelo lógico como um ponto negativo. Para este exemplo, o ganho de desempenho pode vir, por exemplo, da possibilidade de leitura apenas dos dados de um veículo a partir de sua coleção, sem trazer obrigatoriamente todos os dados do usuário e dos demais veículos deste usuário.

3. Coleção com array de documentos embutidos mais coleção com apenas um documento embutido

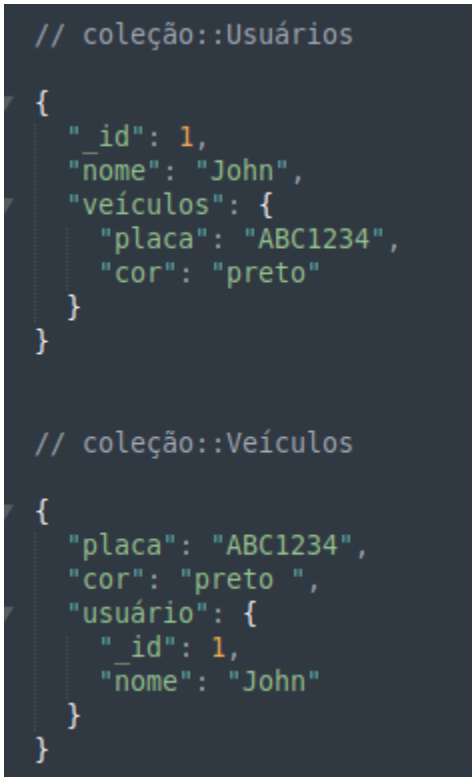
Figura 26 - Relacionamento bidirecional com redundância de documentos das duas entidades



Fonte: De autoria própria

A Figura 26 tem propriedades semelhantes às do primeiro exemplo, pois ela traz consigo os mesmos ganhos de desempenho de leitura - ao tornar possível obter todos os dados de veículo e usuário a partir da leitura de um único documento - com a vantagem de permitir maior flexibilidade de consultas. Essa flexibilidade se dá pois a partir desta redundância de dados podemos realizar consultas tanto na coleção de usuários quanto na coleção de veículos e os dados de ambas as entidades sempre serão retornados. Por outro lado, isso pode se tornar uma desvantagem em casos que a intenção seja obter somente os dados de uma única entidade. Outra desvantagem diz respeito à maior dificuldade de consistência na medida que os dados das entidades estão redundantes e a própria informação da relação também está, sendo necessário tratar sempre de forma conjunta alterações em ambas as entidades para garantir a consistência das mesmas.

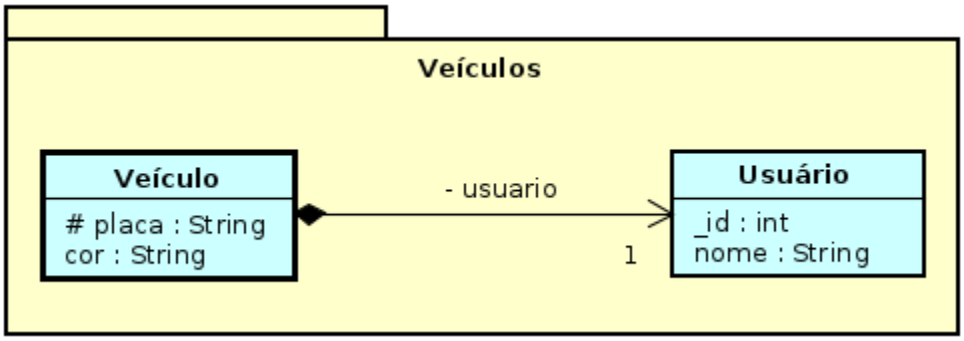
Figura 27 - JSON de relacionamento bidirecional com redundância de documentos das duas entidades em ambos os lados da relação



Fonte: De autoria própria

4. Coleção com apenas um documento embutido

Figura 28 - Inversão de raíz do agregado no modelo de coleção única



Fonte: De autoria própria

Na Figura 28, em comparação à Figura 22, os papéis no agregado se invertem, veículo se torna a raíz do agregado. Em termos de vantagens e desvantagens este modelo preserva as mesmas propriedades que o primeiro exemplo, porém como estrutura JSON são persistidos de maneira diferente.

Figura 29 - Estrutura JSON de agregado único com veículo sendo a raíz

```
// coleção::Veículos

{
  "placa": "ABC1234",
  "cor": "preto ",
  "usuário": {
    "_id": 1,
    "nome": "John"
  }
}
```

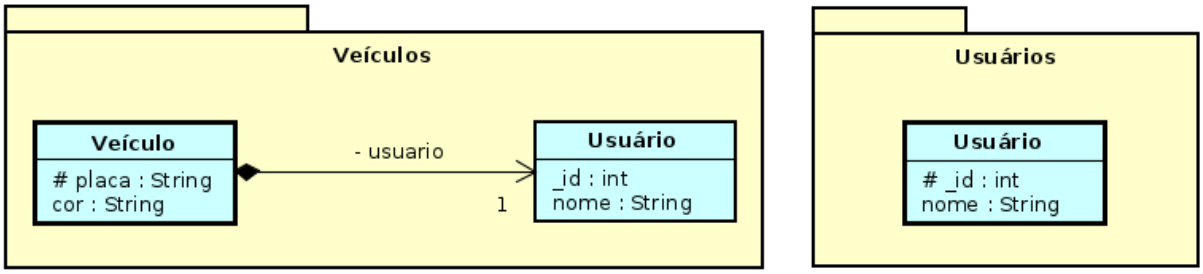
Fonte: De autoria própria

Vaughn Vernon [5] diz que o requisito para se tornar raíz de um agregado é que a entidade tenha uma identidade única global, ou seja, entre todos os agregados. No exemplo em questão o veículo tem uma identidade única: a placa, e que, dependendo do caso de uso pode ser mais interessante que seja utilizada para ser o identificador dos documentos que

algum atributo do usuário. Esta abordagem porém exige uma maior quantidade de leituras na busca por todos os veículos de um usuário, dado que os veículos estão em documentos distintos.

5. Coleção com apenas um documento embutido mais coleção redundante

Figura 30: Usuário como documento embutido de veículo e em coleção própria



Fonte: De autoria própria

A Figura 30 carrega as mesmas características da Figura 24 (exemplo 2), porém a redundância está nos documentos de usuários e não em veículos. Isto é, desta forma podemos realizar buscas e leituras apenas dos usuários se for necessário e garantimos também que as informações dos usuários estão sempre presentes em casos que seja necessário realizar uma busca de veículos. Esse agrupamento evita, ao realizar a busca de um veículo, ter de realizar algum tipo de JOIN para obter os dados dos usuários donos desses veículos.

Como ponto negativo, ao contrário do exemplo 2, uma listagem de veículos por usuário pode ser computacionalmente mais custosa, sendo necessário a leitura de mais de um documento para montar a listagem de veículos. Copeland [9] afirma que essa modelagem em termos de custo computacional é mais custosa, exemplificando o fato que no caso do MongoDB o banco garante apenas que os dados de um único documento são armazenados de forma contígua em disco.

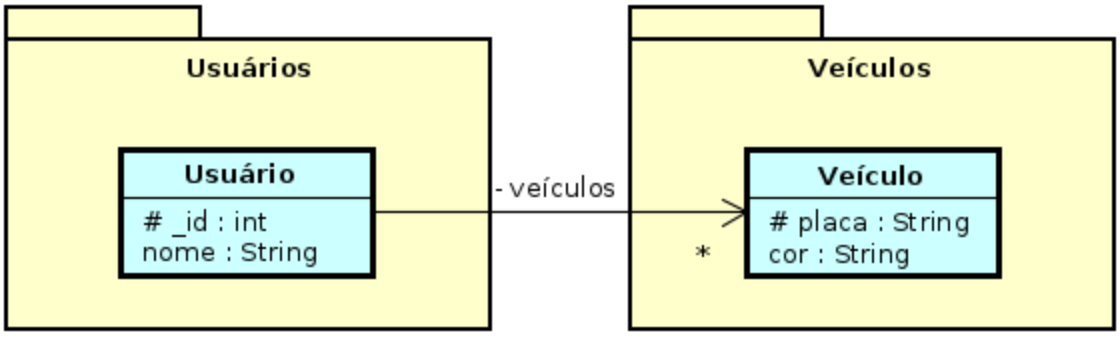
Figura 31 - JSON mostrando modelagem com usuário como documento embutido de veículo e redundante em coleção própria

```
// coleção::Veículos

{
  "placa": "ABC1234",
  "cor": "preto",
  "usuário": {
    "_id": 1,
    "nome": "Pedro",
  }
}


// coleção::Usuários

{
  "_id": 1,
  "nome": "Pedro",
}
```

Fonte: De autoria própria

6. Coleção com array de documentos referenciados

Figura 32 - Veículo como array de referências dos usuários



Fonte: De autoria própria

Com relação aos exemplos de modelagem vistos até aqui, este é o primeiro que utiliza referência no lugar de documentos embutidos para realizar o relacionamento entre usuário e veículos através da notação AML de dois agregados distintos. Copeland [9] discute as vantagens e desvantagens entre as duas abordagens para uma tomada de decisão de qual utilizar. Segundo ele, no relacionamento de documentos embutidos existem ganhos de localidade, ou seja, a garantia de que ambas as entidades permanecem fisicamente juntas na mesma instância de banco. Outro ganho viria através do que ele chama de atomicidade**,** ou seja, existem ganhos de consistência de dados ao editar atributos de ambas as entidades e aplicar essas alterações de forma conjunta. Já no modelo de relacionamento por referência os ganhos viriam em termos de flexibilidade e escalabilidade**.** A flexibilidade se dá pelo fato de permitir, quando necessário, buscas e leituras diretas por qualquer uma das entidades sem necessidade de trazer no resultado também dados da outra entidade. Já em termos de ganho de escala Copeland [9] cita as próprias limitações de tamanho de documentos, no caso do MongoDB, 16MB por documento. Essa limitação no exemplo de relacionamento por documento embutido limitaria a quantidade de veículos relacionados com um usuário a partir desta limitação do tamanho máximo do documento do usuário.

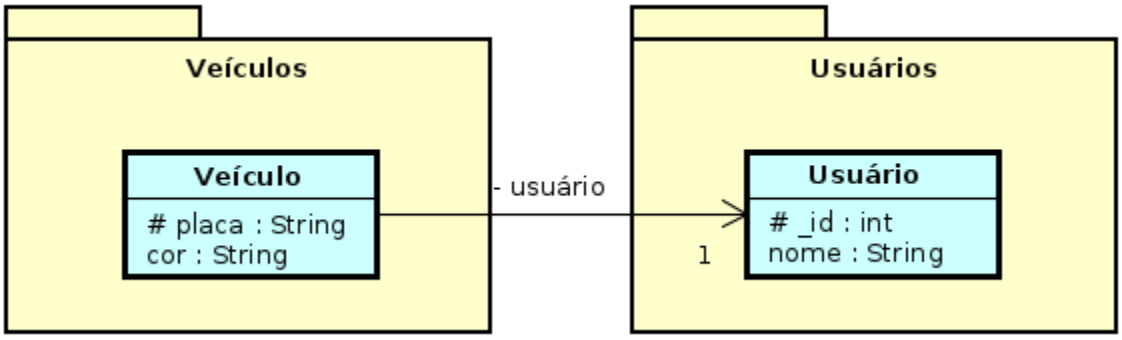
Neste relacionamento seriam gerados dois documentos JSON independentes:

Figura 33 - Relacionamento usuário-veículo através de array de referências em JSON

Fonte: De autoria própria

7. Coleção de documentos referenciados

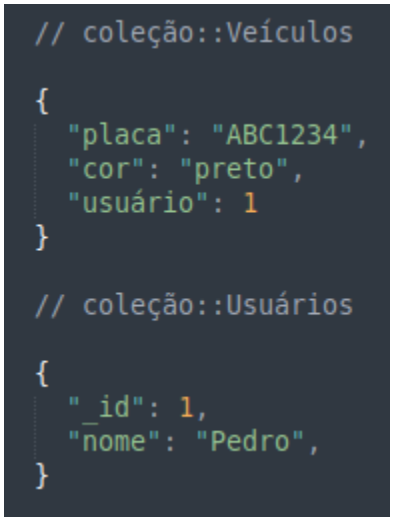
Figura 34 - Usuário como referência de veículos



Fonte: De autoria própria

Em termos de propriedades, a Figura 34 é semelhante à Figura 32 (exemplo 6), porém a escolha de colocar a referência nos veículos no lugar de usuários dependendo do cenário pode trazer algumas vantagens. Sadalage e Fowler [6] dizem que em um cenário ideal os dados de um mesmo usuário deveriam permanecer em uma mesma instância do banco para obter ganhos de desempenho de leitura. A partir desta perspectiva, considerando que a coleção de veículos seja particionada através da funcionalidade de *sharding* do MongoDB, no exemplo anterior apenas com as informações que temos do lado da entidade veículo não é possível ter essa garantia. Na modelagem atual porém a entidade veículo tem o identificador do usuário como é possível ver no JSON de exemplo:

Figura 35 - Relacionamento usuário-veículo através de referência na entidade usuário em JSON

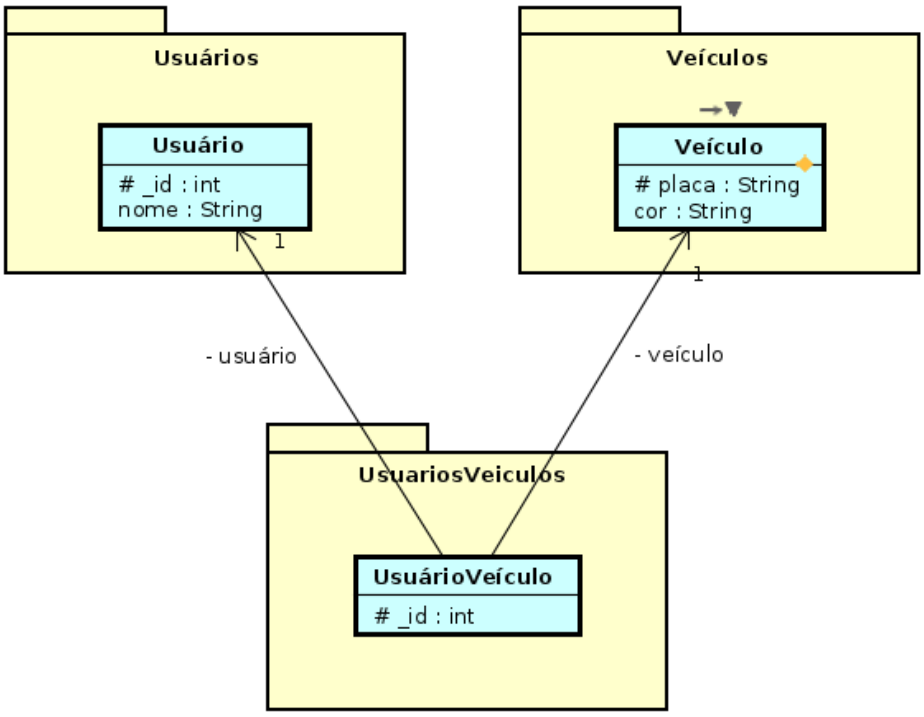


Fonte: De autoria própria

Desta forma ao realizar um particionamento no MongoDB da coleção Veículo é possível definir como chave de particionamento (ou sharding key) o identificador de usuário, permitindo que uma listagem de todos os veículos de determinado usuário possa ser atendida requisitando uma única instância de banco.

8. Coleção intermediária de referências

Figura 36 - Relacionamento usuário-veículo através de referências em entidade de relacionamento



Fonte: De autoria própria

Copeland [9] fala sobre este tipo de relacionamento, citando que embora não haja duplicidade de dados nas entidades Usuário e Veículo ele traz complexidade nas consultas, necessitando da utilização de JOINS, semelhante a Figura 6, pois diferentemente dos documentos embutidos, documentos referenciados exigem junções quando precisam ser consultados em conjunto. Contudo, nesse cenário, tem-se uma junção extra. Ou seja, existe uma perda de desempenho maior sempre que necessário a leitura dos veículos de um usuário, ou de dados de usuário dono de um veículo, logo é necessário pesar o tradeoff desempenho e consistência. Ao eliminar a coleção intermediária e duplicar os dados das entidades usuario dentro dos documentos de veículos e vice-versa eliminamos a necessidade de JOINS, porém temos que ter cuidados para garantir a consistência dessas relações em ambas as coleções. Este prejuízo de consistência não ocorre na modelagem com uma coleção intermediária realizando o relacionamento pois permite eliminar e criar relacionamentos de forma atômica, ou seja, com escrita em um único documento.

Exemplo de coleções e documentos em JSON representando esta modelagem:

Figura 37 - Relacionamento usuário-veículo através de referências em entidade intermediária

```
// coleção::Usuários

{
  "_id": 1,
  "nome": "Pedro",
}

// coleção::Veículos

{
  "placa": "ABC1234",
  "cor": "preto",
}

// coleção::UsuáriosVeículos

{
  "_id": 1,
  "usuário": 1,
  "veículo": "ABC1234"
}
```

Fonte: De autoria própria

Analisando o JSON da coleção UsuárioVeículo na Figura 37, é possível identificar uma outra utilização deste tipo de modelagem. Em um cenário que seja permitido realizar transferências de veículos entre usuários é possível manter um histórico destes vínculos na entidade intermediária e adicionar um campo extra sinalizando se o vínculo ainda é válido. Embora este tipo de histórico também possa ser feito nas próprias coleções de Usuário e Veículo pode esbarrar nos limites de tamanho de documento como já discutido anteriormente.

4 Conclusão

A Linguagem de Modelagem Lógica AML (Agregate Modeling Language) emerge como uma ferramenta inovadora e valiosa para abordar os desafios de modelagem de bancos de dados orientados a documentos. A capacidade de aproveitar ferramentas de modelagem UML existentes amplia a acessibilidade na adoção da AML, proporcionando uma transição suave para profissionais que já estão familiarizados com a UML. Além disso, a forte base no conceito de agregados do Domain-Driven Design (DDD) promove uma abordagem lógica e coesa na modelagem de dados, alinhando-se com as melhores práticas da indústria.

O presente trabalho mostrou como utilizar os construtores da AML no processo de modelagem lógica de um banco orientado a documentos e explorou a versatilidade da AML ao demonstrar sua capacidade de lidar com variações distintas na modelagem de relacionamentos de um esquema conceitual específico. Também foram discutidos os impactos de desempenho e consistência ao escolher determinada modelagem de relacionamento em detrimento de outra.

A partir dos exemplos de modelagem mostrados na Seção 3, é possível utilizar este trabalho uma fonte de consulta para modelagens mais genéricas em casos que se fizer necessário considerar o tradeoff entre consistência e desempenho na modelagem dos relacionamentos independente do domínio em questão.

O fato de ter utilizado o MongoDB como referência de implementação de banco orientado a documentos bem como estrutura JSON utilizada pelo mesmo, por outro lado, pode limitar a aplicação dos exemplos mostrados em outros contextos de bancos e estruturas de documento distintas.

Em um cenário em constante evolução da tecnologia da informação, a AML emerge como uma ferramenta que não apenas facilita a modelagem de sistemas de banco de dados, mas também contribui para a construção de sistemas mais robustos, eficientes e alinhados com as demandas do mundo contemporâneo de bancos de dados orientados a documentos.

5 Bibliografia

[1] EVANS, Eric. *Domain-driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley Professional*, 2004.

[2] KAUFMAN, Michael; MEIER, Andreas. *SQL and NoSQL Databases: Modeling, Languages, Security and Architectures for Big Data Management - Second Edition*. Springer, 2023

[3] FIORI, Alessandro. *Design with MongoDB: Best models for applications*, Publicação própria ISBN: 9798557417884, 2020

[4] Ranking of the most popular database management systems worldwide, as of February 2023 - statista.com - acesso em 10/09/2023

[5] VERNON, Vaughn. *Implementing Domain-Driven Design.* Addison-Wesley Professional, 2013

[6] SADALAGE, Pramod J; FOWLER, Martin. *NoSQL Distilled.* Addison-Wesley Professional, 2012

[7] BUGIOTTI1, Francesca; *Database Design for NoSQL Systems,* Universita Roma Tre, 2014

[8] LIMA, C. *Projeto Lógico de Bancos de Dados NOSQL Documentos a Partir de Esquemas Conceituais Entidade-Relacionamento Estandido(EER)*. Universidade Federal de Santa Catarina, 2016.

[9] COPELAND, Rick. *MongoDB Applied Design Patterns.*

[10] Persistence with NoSQL Databases - DDD - https://www.aschommer.de/blog/persistence-with-nosql-databases-ddd.html - acesso em 15/09/2023

[11] de Lima, C. and dos Santos Mello, R. (2015). *A workload-driven logical design approach for nosql document databases*. In Proceedings of the 17th International Conference on Information Integration and Web-based Applications & Services, pages 1-10

[12] Hoberman, S. *Data modeling for MongoDB: Building well-designed and supportable MongoDB databases (1st ed.) Basking Ridge*, NJ: Technics Publications, 2014