

MongoDB: Modelação de dados

Agenda

- Padrões de modelação Mongo:
 - Embedded Document Pattern
 - Subset pattern
- Processo de modelação
- Tipos de Relacionamento

- Em bases de dados orientadas por documentos, representamos **entidades** como **documentos**;
- No MongoDB, os relacionamentos entre entidades podem ser representados pela técnica de **embutir documentos** ou por **referências**;
- Antes de proceder à construção da base de dados é necessário identificar as principais **entidades** que pretendemos descrever e como estes se **relacionam**;

- O principal desafio do processo de **modelação** de dados passa por **equilibrar** as **necessidades** aplicacionais, as características de **desempenho** da base de dados e os padrões de **recuperação** de dados;
- Ao projetar o modelo de dados, devemos ter em consideração a forma como os dados serão **utilizados** pelas aplicações (ou seja, **consultas**, **atualizações** e processamento dos dados), bem como a **estrutura** inerente dos próprios dados.

Embutir documentos

- Ao **embutir** documentos (*Embedded Document Pattern*) podemos concentrar todo o conteúdo num único documento:

```
user document:
{
  _id: <User1>,
  username: "johndoe",
  firstname: "john",
  lastname: "doe"
  address: {
    city: "Zürich",
    country: "Switzerland"
  },
  contact: {
    phone: "+41 123-123-123",
    email: "jd@jd.com"
  }
}
```

Embedded Address

Embedded contact

Embutir documentos

- Exemplo: Representar as várias **moradas** (addresses): rua, cidade e país) para cada **pessoa** (person): nome, NISS:

Pessoa

```
{
  name: 'Kate Monster',
  ssn: '123-456-7890',
  addresses : [
    { street: '123 Sesame St', city: 'Anytown', cc: 'USA' },
    { street: '123 Avenue Q', city: 'New York', cc: 'USA' }
  ]
}
```

Morada

Embutir documentos

- A grande vantagem da abordagem anterior é que **não** é necessário elaborar **duas consultas** separadas para devolver pessoas e moradas;
- A grande **desvantagem** é que não podemos relacionar a mesma **morada** com mais do que uma pessoa (sem a replicar);
- Imagine o seguinte problema: uma cidade mudou de nome. Quantas operações de **atualização** são necessárias?

Embutir documentos

- Incorporar dados relacionados num único documento pode reduzir o número de operações de leitura necessárias para obtenção de dados;
- Em geral, devemos estruturar os documentos para que as aplicações que o utilizam recebam todas as informações necessárias numa única operação de leitura;

Embutir documentos

- No entanto, a utilização de documentos embutidos pode levar a documentos **de grande dimensão** que contêm campos dos quais as aplicações tipicamente não necessitam;
- Os dados desnecessários podem causar uma **carga extra** e desacelerar as operações de leitura;

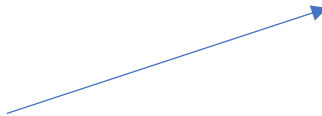
Subset Pattern

- Para mitigar o problema, podemos utilizar o padrão: **Subset** para recuperar o **subconjunto** de dados que é acedido com mais **frequência** numa **única** operação de leitura;
- Ao invés de armazenar todos os dados numa única coleção, podemos **dividir** a coleção em **duas novas** coleções (**Subset Pattern**);

Subset Pattern

- Podemos utilizar o padrão do **subconjunto** para aceder apenas aos dados **exigidos**, ao invés de todo o conjunto de dados incorporados.

Apenas mais recentes



```
{
  "_id": 1,
  "name": "Super Widget",
  "description": "This is the most useful item in your toolbox.",
  "price": { "value": NumberDecimal("119.99"), "currency": "USD" },
  "reviews": [
    {
      "review_id": 786,
      "review_author": "Kristina",
      "review_text": "This is indeed an amazing widget.",
      "published_date": ISODate("2019-02-18")
    },
    {
      "review_id": 785,
      "review_author": "Trina",
      "review_text": "Nice product. Slow shipping.",
      "published_date": ISODate("2019-02-17")
    },
    ...
    {
      "review_id": 1,
      "review_author": "Hans",
      "review_text": "Meh, it's okay.",
      "published_date": ISODate("2017-12-06")
    }
  ]
}
```

Subset Pattern

- As reviews são ordenadas em ordem cronológica inversa. Quando um utilizador visita uma página de um produto, a aplicação carrega as dez reviews mais recentes.
- Por isso,, podemos optar por armazenar as mais antigas numa coleção separada e incorporar as mais recentes.

Subset Pattern

```
{
  "review_id": 786,
  "product_id": 1,
  "review_author": "Kristina",
  "review_text": "This is indeed an amazing widget.",
  "published_date": ISODate("2019-02-18")
}
{
  "review_id": 785,
  "product_id": 1,
  "review_author": "Trina",
  "review_text": "Nice product. Slow shipping.",
  "published_date": ISODate("2019-02-17")
}
...
{
  "review_id": 1,
  "product_id": 1,
  "review_author": "Hans",
  "review_text": "Meh, it's okay.",
  "published_date": ISODate("2017-12-06")
}
```

Cada análise contém uma referência ao produto para o qual foi escrita.

Subset Pattern

- A utilização de documentos mais pequenos contendo dados de acesso mais frequentes reduz o tamanho global do conjunto de trabalho.
- Estes documentos mais pequenos resultam num melhor desempenho de leitura para os dados a que a aplicação acede com mais frequência.

Subset Pattern

- No entanto, o padrão do subconjunto resulta em **duplicação** de dados.
- No exemplo anterior, as reviews são mantidas na coleção de produtos e na coleção de reviews.
- Devem ser tomadas medidas adicionais para assegurar que as reviews sejam **consistentes** entre cada coleção.

Múltiplas Coleções

- Existem cenários em que ter colecções **separadas** faz sentido.
- Se uma entidade pode ser pensada como uma entidade **separada** e **independente**, muitas vezes faz sentido ter uma coleção separada;
- Nestes casos, podemos otimizar a estrutura utilizando o padrão: **Subset** discutido anteriormente.

Múltiplas Coleções - Exemplo

Order Collection

```
{
  _id: ObjectId("507f1f77bcf86cd799439011"),
  date: ISODate("2019-02-18"),
  customer_id: 123,
  order: [
    {
      product: "widget",
      qty: 5,
      cost: {
        value: NumberDecimal("11.99"),
        currency: "USD"
      }
    }
  ]
}
```

Customer Collection

```
{
  _id: 123,
  name: "Katrina Pope",
  street: "123 Main St",
  city: "Somewhere",
  country: "Someplace",
  ...
}
```

Inventory Collection

```
{
  _id: ObjectId("507f1f77bcf86cd111111111"),
  name: "widget",
  cost: {
    value: NumberDecimal("11.99"),
    currency: "USD"
  },
  on_hand: 98325,
  ...
}
```

Múltiplas Coleções

- De um ponto de vista de **desempenho** torna-se problemático, uma vez que precisamos de “juntar as peças” para uma ordem específica.
- Um cliente pode ter N encomendas, criando um relacionamento 1-N;
- Incorporar toda a informação sobre um cliente para cada encomenda apenas para reduzir as operações de Junção (**lookup**) resulta numa grande quantidade de informação **duplicada**.
- Além disso, nem toda a informação sobre um cliente pode ser **necessária** para uma encomenda.

Múltiplas Coleções

- O padrão: [Extended Reference](#), proporciona uma ótima forma de lidar com estas situações;
- Ao invés de duplicarmos toda a informação sobre o cliente, apenas duplicamos os campos a que temos acesso com [frequência/prioridade](#).

Múltiplas Coleções

Customer Collection

```
{
  _id: 123,
  name: "Katrina Pope",
  street: "123 Main St",
  city: "Somewhere",
  country: "Someplace",
  date_of_birth: ISODate("1992-11-03"),
  social_handles: [
    twitter: "@somethingamazing123"
  ]
  ...
}
```

Order Collection

```
{
  _id: ObjectId("507f1f77bcf86cd799439011"),
  date: ISODate("2019-02-18"),
  customer_id: 123,
  shipping_address: {
    name: "Katrina Pope",
    street: "123 Main St",
    city: "Somewhere",
    country: "Someplace"
  },
  order: [
    {
      product: "widget",
      qty: 5,
      cost: {
        value: NumberDecimal("11.99"),
        currency: "USD"
      }
    }
  ],
  ...
}
```

Múltiplas Coleções

- Algo a considerar na aplicação neste padrão são os dados **duplicados**;
- Portanto, funciona melhor se os dados que são armazenados no documento principal forem campos que **não mudam frequentemente**.

Relacionamentos entre documentos

- Temos diferentes tipos de relacionamentos:
 - 1:1
 - 1:N
 - M:N

Relacionamentos 1:1

- “Um utilizador tem uma morada e uma morada está associada a um utilizador”
- Tipicamente embutimos os documentos.

Relacionamento 1:N

- “Um veículo possui várias peças e uma peça está associada a um veículo”
- Considere: Não sabemos ao certo **quantas** peças cada veículo terá, podem ser algumas dezenas mas também podem ser algumas **centenas** (tendo sempre um limite expectável);
- Tipicamente criamos um modelo de dados que use documentos **embutidos** para descrever um relacionamento de um para muitos entre dados conectados.
- No entanto, temos de ter em consideração se existem padrões de **recuperação** apenas para **peças**, o que poderá implicar uma coleção **separada** só para as peças e a aplicação do padrão: *Extended Reference*.

Relacionamento 1:N

- Considere que se pretende modelar **Editoras** e **Livros**.
- A incorporação do documento da editora dentro do documento do livro levaria à **repetição** dos dados da editora.
- Para evitar a repetição dos dados da editora, podemos utilizar referências e manter as informações da editora numa **coleção separada** da coleção de livros (desde que os **requisitos de recuperação** o justifiquem).
- Ao usar referências, o crescimento dos relacionamentos determina **onde armazenar** a referência.

- “Um Livro tem vários autores e um autor tem vários livros”
- Depende dos padrões de **recuperação**. Especificamente para a associação entre autores e livros:
 - Se as consultas forem essencialmente por **livro**:

```
{
  _id: "book001",
  title: "Cell Biology",
  authors: [
    {
      author_id: "author124",
      name: "Ellie Smith"
    },
    {
      author_id: "author381",
      name: "John Palmer"
    }
  ]
}
```

- Se as consultas forem essencialmente por **autor**:
- No primeiro caso poderia existir uma coleção separada para autores e para o segundo caso uma coleção de livros (**Extended Reference Pattern**)

```
{
  _id: "author124",
  name: "Ellie Smith",
  books: [
    {
      _id: "book001",
      title: "Cell Biology",
    },
    {
      _id: "book002",
      title: "Anatomy and Physiology",
    }
  ]
}
```

Computed Pattern

- Existem outros padrões úteis: <https://www.mongodb.com/blog/post/building-with-patterns-a-summary>.
- O **Computed Pattern** é utilizado quando temos dados que precisam ser computados repetidamente em nossa aplicação.

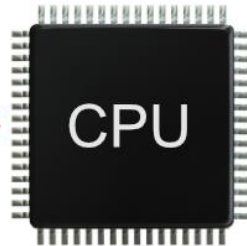
Computed Pattern

Screening Information

```
{  
  "ts": DateTime(XXX),  
  "theater": "Alger Cinema",  
  "location": "Lakeview, OR",  
  "movie_title": "Jack Ryan: Shadow Recruit",  
  "num_viewers": 344,  
  "revenue": 3440  
}
```

```
{  
  "ts": DateTime(XXX),  
  "theater": "City Cinema",  
  "location": "New York, NY",  
  "movie_title": "Jack Ryan: Shadow Recruit",  
  "num_viewers": 1496,  
  "revenue": 22440  
}
```

```
{  
  "ts": DateTime(XXX),  
  "theater": "Overland Park Cinema",  
  "location": "Boise, ID",  
  "movie_title": "Jack Ryan: Shadow Recruit",  
  "num_viewers": 760,  
  "revenue": 7600  
}
```



Movie Information

```
{  
  "ts": DateTime(XXX),  
  "title": "Jack Ryan: Shadow Recruit",  
  "viewers": 2600,  
  "revenue": 33480  
}
```

- Tarefas:
 - Identificar as **consultas** (padrões de recuperação) que executa com mais frequência
 - Identificar os relacionamentos nos dados para decidir se utilizamos referências ou aplicamos a técnica de embutir documentos.
 - Aplicar **padrões** de design de esquema para otimizar leituras e gravações
 - Criar **índices** para suportar padrões de query comuns.

Processo de design – Identificar consultas

- Considerar os cenários atuais e os cenários futuros.
- Construir uma tabela:

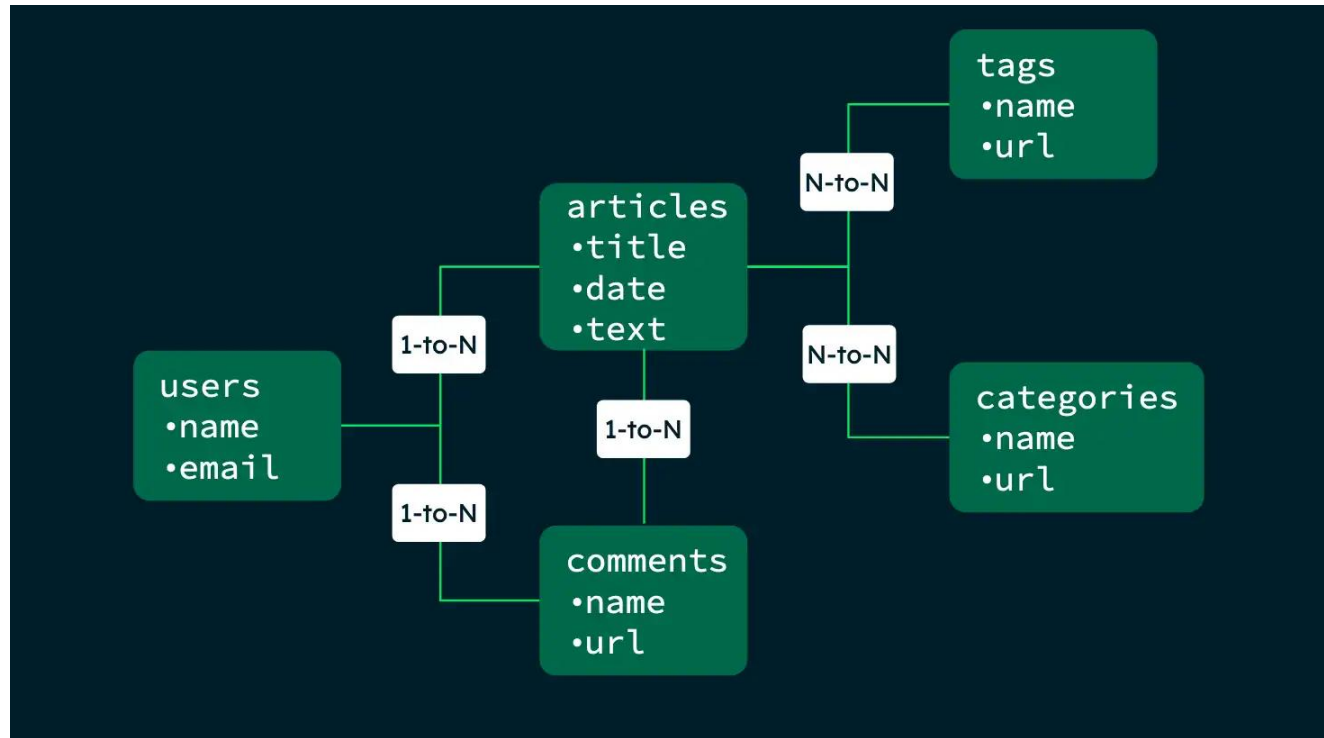
Ação	Tipo	Dados	Frequência	Priority
Enviar um novo artigo	Escrita	autor, texto	10 por dia	Alta
Enviar um comentário sobre um artigo	Escrita	Utilizador, texto	1,000 per day (100 per article)	Médio
Ver um artigo	Leitura	ID do artigo, texto, comentários	1,000,000 por dia	Alta
Ver análise de artigo	Leitura	ID do artigo, comentários, cliques	10 por hora	Baixo

Processo de design – Identificar Relacionamentos

- Para determinar se devemos **embutir** ou utilizar **referências**:
 - Se existem consultas que frequentemente devolvem dados de uma **entidade** para devolver dados sobre outra **entidade**, **embutimos** os dados para evitar a necessidade **lookups**.
 - Se as consultas **atualizam** frequentemente os dados **relacionados**, considere armazenar os dados na sua **própria coleção** e usar uma **referência**. Ao utilizar uma referência, reduzimos a carga de trabalho de gravação, uma vez que só atualizamos os dados em um único local.

Processo de design

- Identificar relacionamentos



Processo de design – Aplicar Padrões

- Utilizar **padrões de design** para otimizar o modelo de dados com base nos padrões de recuperação.
- Os padrões de design de esquemas melhoram o desempenho do aplicativo e **reduzem a complexidade** do esquema.

Processo de design – Índices

- Um índice abrange uma consulta quando o **índice** contém os campos utilizados pela query.
- Uma coleção pode ter um máximo de **64** índices.
- Para cenários com altas taxas de **inserção**, os índices podem **degradar** o desempenho porque cada inserção também deve atualizar índices.

Manipular dados duplicados

- Uma preocupação com a duplicação de dados é o aumento dos custos de **armazenamento**.
- No entanto, os **benefícios** da otimização dos padrões geralmente superam os possíveis aumentos de custos do armazenamento.

Manipular dados duplicados

- Antes de duplicar dados, considere os seguintes fatores:
 - Com que **frequência** os dados duplicados precisam ser **atualizados**? A atualização frequente de dados duplicados pode **causar cargas de trabalho pesadas** e problemas de desempenho.
 - No entanto, a **lógica extra** necessária para lidar com atualizações pouco frequentes é **menos dispendiosa** do que a execução de junções (pesquisas) em operações de leitura.
 - O **benefício** em desempenho das **leituras** quando os dados estiverem duplicados. A duplicação de dados pode eliminar a necessidade de realizar **lookups** em várias coleções, o que pode **melhorar** o **desempenho** do aplicativo.

- **Embutir** o máximo possível: a BD do documento deve eliminar bastante as junções de documentos e por isso devemos colocar o máximo possível de informação num único documento.
- **Guardar** e **recuperar** um documento atómico é bastante rápido.

- **Normalizar** os dados que podem ser referenciados em múltiplas entidades ou possuem um número de instâncias indeterminado;
- Isto significa que devemos criar coleções **reutilizáveis** (por exemplo, país ou utilizador);
- Esta é a forma mais **eficiente** de lidar com valores duplicados.

- O MongoDB pode armazenar documentos com estruturas de dados **arbitrárias** e profundas, mas não pode efetuar pesquisas de forma eficiente em todos os cenários;
- Se os dados formam uma árvore extensa, deve ser ponderada a estratégia de armazenar dados em documentos **separados**;
- Consistência: a eficiência de MongoDB implica como consequência a **consistência** dos dados.

- No processo de **modelação**, as seguintes questões devem ser ainda consideradas:
 - As entidades do lado do **N** (1:N) necessitam de existir **isoladamente**?
 - É uma questão que **depende** do problema. As moradas poderiam existir isoladamente, mas as reviews só existem se existirem restaurantes;
- Considerar a aplicação de padrões de **design**.

- Qual é a cardinalidade do relacionamento 1:N?
 - 1 “para poucos”
 - 1 “para muitos”
 - 1 “para um número indeterminado”

- Referências Web:

- <https://www.json.org/json-en.html>
- <https://docs.mongodb.com/manual/>
- <https://docs.mongodb.com/manual/core/data-modeling-introduction/>
- <https://docs.mongodb.com/manual/applications/data-models/>
- Hoberman, S., 2014. Data-Modeling for MongoDB-Building Well Designed and Supportable MongoDB Databases, frist. ed. Technics Publications, LLC.

MongoDB: Modelação de dados