Banco de Dados NoSQL

Aula 6: Modelo de dados NoSQL orientado a documentos – parte II

Apresentação

Na Aula 5, iniciamos o estudo do modelo de banco de dados NOSQL orientado a documentos, onde executamos comandos para a definição da estrutura de tabelas no sistema gerenciador de banco de dados MongoDB usando a MQL (MongoDB Query Language). Foram escritas ainda consultas para a realização de operações básicas de cadastro, pesquisa , atualização e exclusão de dados. Porém, não foram apresentados exemplos com modelos de dados agregados, de forma comparativa ao modelo entidade-relacionamento. Assim, continuaremos estudando o banco de dados orientado a documentos, criando estudos de casos envolvendo coleções. E faremos também um comparativo das implementações no MongoDB e no PostgreSQL, usando o JSONB.

Objetivo

- Reconhecer os modelos de dados agregados;
- Analisar estudos de casos de modelos de dados agregados de documentos;
- Escrever comandos NoSQL para a manipulação de dados em modelos de dados agregados de documentos.

Introdução

Na Aula 5, foi iniciado o estudo do banco de dados NOSQL no modelo orientado a documentos (SADALAGE; FOWLER, 2013), que também utiliza pares de chaves-valor, com ouso de coleções. Porém, sabemos que um banco de dados é composto por muitas tabelas (no modelo relacional) e coleções (no modelo orientado a documentos). Por isso, é necessário o estudo de como projetar o banco de dados de forma a melhor armazenar e atender às necessidades de consultas de seus usuários.

Dessa forma, antes de iniciarmos o estudo de como projetar bancos de dados NOSQL orientado a documentos, devemos entender o conceito de agregados. Após isso,e por meio de estudos de casos, vamos comparar o projeto no modelo relacional com o modelo de dados agregados para a construção de um banco de dados orientado a documentos. No final da aula, vamos criar os projetos no MongoDB e no PostgreSQL para facilitar a comparação e a identificação das diferenças nos projetos envolvendo esses dois modelos de bancos de dados.

Modelo de dados agregados

Em termos de modelagem, a principal diferença entre os modelos relacional e orientado a documentos é que a modelagem do primeiro é baseada no modelo entidade-relacionamento(ELMASRI; NAVATHE, 2018),e a modelagem das categorias chave-valor orientadas a documentos e orientadas a colunas seguem o modelo de dados agregados. No modelo relacional, os registros armazenam dados de apenas uma entidade (exceção das tabelas de associação). Na modelagem com agregados, os dados são definidos em unidades e essas unidades são estruturas menos complexas e que facilitam o trabalho em determinadas aplicações.

O termo “agregado” foi criado no surgimento do projeto orientado a domínios (domain-driven design)(SADALAGE; FOWLER, 2013). O agregado é um conjunto de objetos relacionados e usados para facilitar o desenvolvimento do funcionamento de aplicações em clusters. Isso ocorre porque o agregado pode ser compreendido como uma unidade natural que

permite melhor tanto a replicação quanto a fragmentação dos dados. Essa ideia de unidade facilita ainda o trabalho da implementação de software por parte dos desenvolvedores.

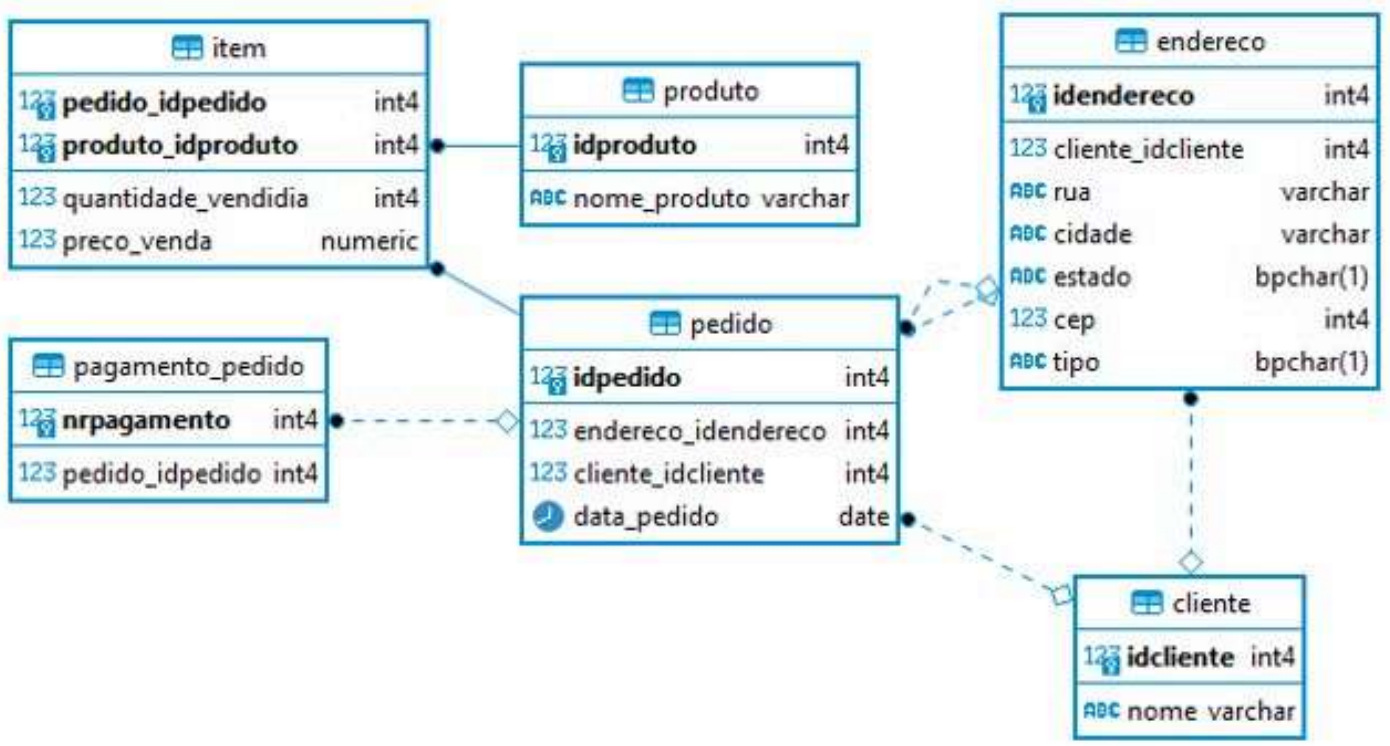
Estudos de casos de modelos de dados agregados de documentos

Para iniciar os estudos de casos comparativos, verificaremos o diagrama entidade-relacionamento apresentado na Figura 1 para o controle de pedidos de clientes. O esquema foi criado para um banco de dados relacional e possui seis tabelas, sendo duas delas tabelas de associação. O cliente tem uma lista de endereços de cobrança e um endereço de envio e seus pagamentos, e o pedido contém uma lista de itens solicitados. O pagamento tem um endereço de cobrança. A partir desse esquema, serão apresentados dois modelos com agregações para uso no modelo orientado a documentos. Existem dois relacionamentos entre pedido e endereço, um indicando o endereço de remessa e outro o de cobrança.

Comentário

As tabelas apresentam um número limitado de atributos por questão de espaço!

Figura 1 - Diagrama entidade-relacionamento de um esquema para o controle de pedidos de clientes para uso no modelo relacional

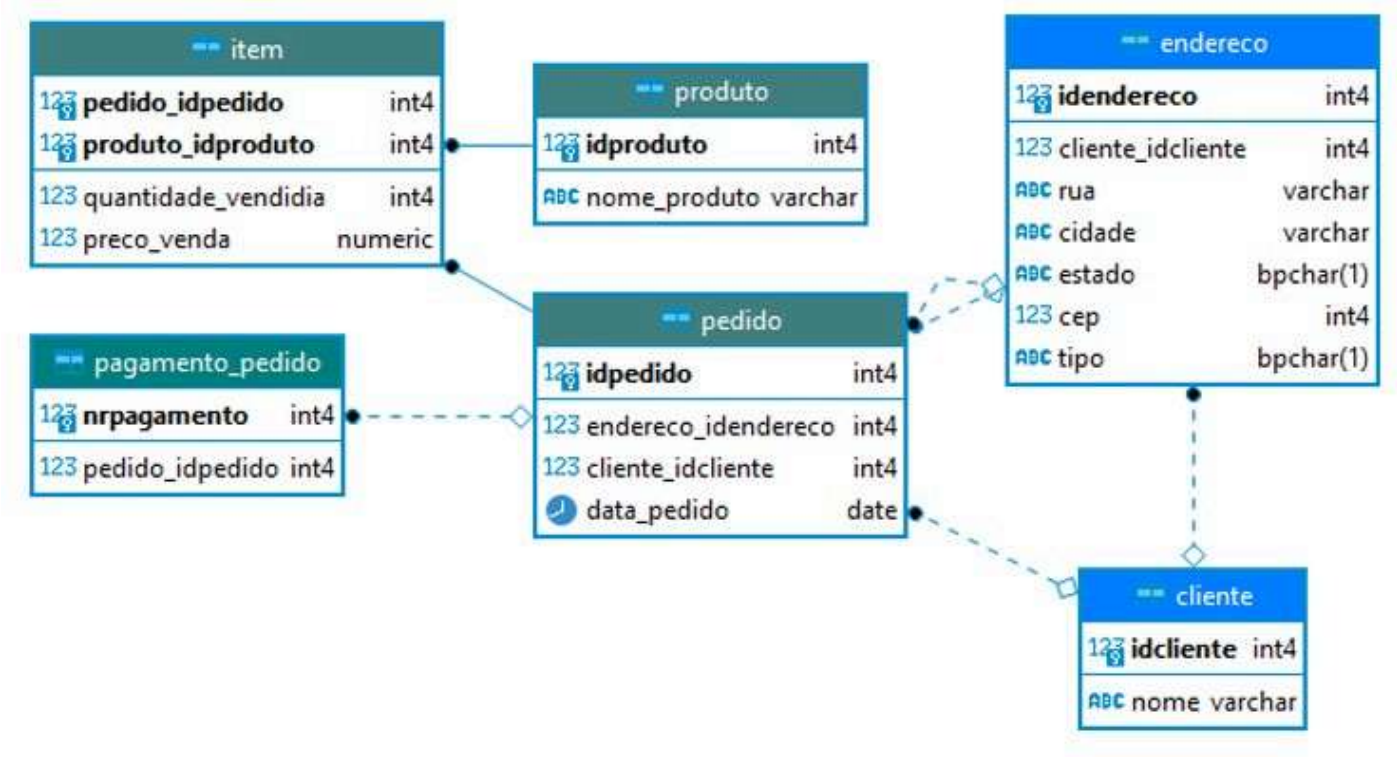


Fonte: O autor.

Para a definição dos agregados deve-se considerar o conjunto de dados que normalmente são acessados pelos usuários e que permitem responder às consultas mais usadas por eles; tudo isso para possibilitar uma forma eficiente de desempenho da aplicação. O primeiro modelo propostopara a transformação do projeto no modelo relacional para o orientado a documentos tem dois agregados principaiscompostos pelos dados do cliente (em azul)e outro com os dados de seus pedidos e dos produtos adquiridos (em verde).

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Figura 2 - Visão dos dois agregados definidos para o projeto de controle de pedidos de clientes



Fonte: O autor.

Comandos NoSQL para manipulação de dados em modelos de dados agregados de documentos

Apresentaremos a seguir os códigos para esses agregados em MQL, a linguagem de consultas do MongoDB.Iniciando com o comando para a criação do banco de dados controle_pedidos:

```
use controle_pedidos
```

Após a criação do banco, são criadas as duas coleções: cliente e pedido, correspondentes aos dois agregados, e depois as coleções são exibidas com o comando show:

```
db.createCollection("cliente")
db.createCollection("pedido")
show collections
```

Com a tabela criada, é apresentado a seguir o comando para a inclusão de um registro contendo os dados de um cliente com os seus endereços de cobrança e entrega:

```
db.cliente.insert({
idcliente : 1 ,
nome : 'Ana' ,
endereco_cobranca : { rua :'A', cidade :'Petrópolis', estado :'RJ', CEP :'20000-000', tipo :'cobrança' } ,
```

```
endereco_remessa : [ { nr_endereco_remessa : 1, rua :'B', cidade : 'Rio de Janeiro' , estado : 'RJ', CEP
:'25000-000' } ,
{ nr_endereco_remessa : 2, rua :'C', cidade : 'Rio de Janeiro' , estado : 'RJ', CEP :'26000-000' } ]});
```

Agora, vamos executar os comandos a seguirpara incluir os dados desses pedidos e verificar como ficam estruturados os documentos na coleção pedido:

```
db.pedido.insert (
{ idcliente : 1,
nr_endereco_remessa :1,
idpedido :1,
data_pedido:newDate("<2020-10-06>"),
item :
    [ {idproduto : 10 , nome_produto : 'produto_1',
    quantidade_vendida : 10 , preco_venda : 25.50 , nrpagamento : 10 } ,
       {idproduto :11 , nome_produto : 'produto_2' ,
    quantidade_vendida : 20 , preco_venda : 35.00 , nrpagamento : 15 } ] } ,
{ idcliente : 1 ,
nr_endereco_remessa :2 ,
idpedido :2 ,
data_pedido :newDate("<2020-10-07>"),
    item : [ { idproduto : 10 , nome_produto : 'produto_3',
    quantidade_vendida : 15 , preco_venda : 5.50 , nrpagamento : 101 } ,
       {idproduto :11 , nome_produto : 'produto_4',
    quantidade_vendida : 40 , preco_venda : 5.00 , nrpagamento : 150 } ] });
```

Para verificar o conteúdo das tabelas criadas, pode-se executar os seguintes comandos:

```
db.cliente.find()
db.pedido.find()
```

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

A solução indicada para aplicações que necessitam do conjunto de dados em apenas um agregado é apresentada a seguir:

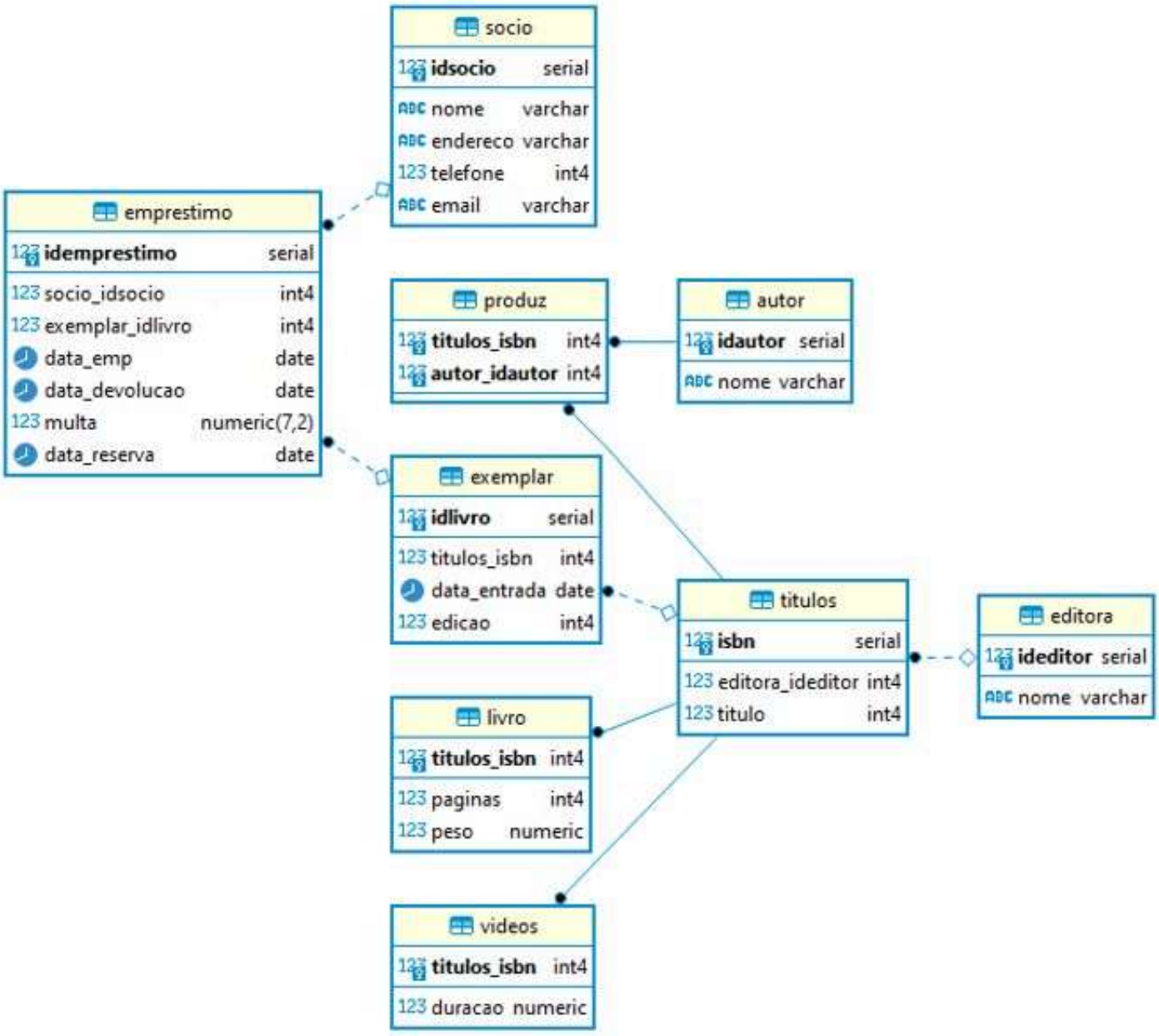
```
db.cliente_pedidos.insert (
{ idcliente : 1 ,
nome :'Ana',
endereco_cobranca : { rua : 'A', cidade : 'Petrópolis', estado : 'RJ', CEP : '20000-000' , tipo : 'cobrança' } ,
endereco_remessa : [ { nr_endereco_remessa : 1, rua : 'B', cidade : 'Rio de Janeiro' , estado : 'RJ', CEP
:'25000-000' } ,
{ nr_endereco_remessa : 2 , rua :'C' , cidade : 'Rio de Janeiro' , estado : 'RJ', CEP : '26000-000' } ] ,
```

```
idpedido :1 ,
    data_pedido :newDate("<2020-10-06>") ,
item : [ { idproduto : 10 , nome_produto : 'produto_1' , quantidade_vendida : 10 , preco_venda : 25.50 ,
nrpagamento : 10 } ,
{ idproduto : 11 , nome_produto : 'produto_2' , quantidade_vendida : 20 , preco_venda : 35.00 , nrpagamento : 15
}
       ] } ,
{
    idpedido :2 ,
    data_pedido :newDate("<2020-10-07>") ,
item : [ { idproduto : 10 , nome_produto : 'produto_3' , quantidade_vendida : 15 , preco_venda : 5.50 ,
nrpagamento : 101 } ,
{ idproduto : 11 , nome_produto : 'produto_4' , quantidade_vendida : 40 , preco_venda : 5.00 , nrpagamento : 150
}
       ] });
```

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

A seguir, será apresentado um estudo de caso sobre o controle de empréstimos de bibliotecas. Esse projeto, porém, será implementado no PostgreSQL, no qual os documentos serão conjuntos de registros com o atributo JSONB. Essa é uma alternativa que apresenta como vantagem a possibilidade de trabalhar com o uso do projeto relacional e do orientado a documentos em um mesmo banco de dados. A Figura 3, a seguir, apresenta como ficaria o projeto implementado no modelo relacional, de modo a permitir a posterior comparação com o projeto orientado a documentos com agregados.

Figura 3 – Diagrama entidade-relacionamento para o controle de empréstimos de uma biblioteca

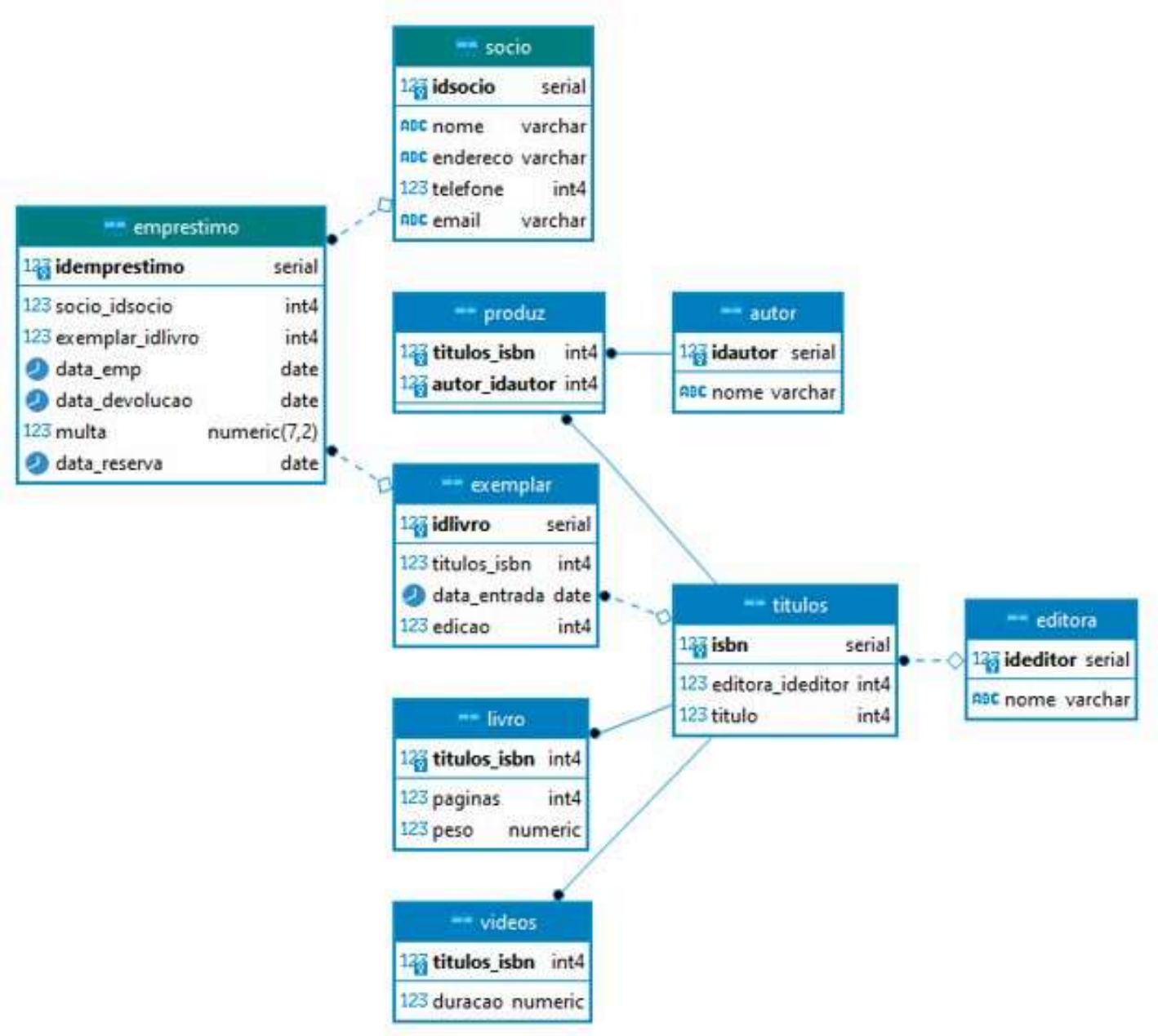


Fonte: O autor.

A Figura 4 apresenta o destaque em verde das tabelas (coleções) para o agregado com os dados do empréstimo e do sócio que o feze, em azul, as tabelas com os dados do acervo.

Logo em seguida, também para permitir a comparação das implementações do projeto nos dois modelos, é apresentado o código SQL somente para a criação das tabelas em um banco de dados relacional.

Figura 4 - Projeto do banco de dados de empréstimos de uma biblioteca com destaque nas visões em cores para os agregados definidos



Fonte: O autor.

A seguir, é apresentado outro estudo de caso para o controle de locações de filmes. Neste estudo de caso, a criação das tabelas será realizada com o código em SQL no PostgreSQL, com o uso de atributos do tipo JSONB. Basicamente foram criadas as tabelas cliente e locação. As tabelas estão relacionadas com uma chave estrangeira indicando as locações que foram realizadas para um determinado cliente. Primeiramente, criaremos a tabela cliente com um atributo JSONB. A seguirsão apresentados os códigos para a criação da tabela cliente e a inclusão dos registros (você não precisar criar essas tabelas, pois elas servem apenas como um comparativo de um projeto no modelo relacional e no modelo orientado a documentos).

```
CREATE TABLE socio (
idsocioSERIAL NOT NULL ,
nome VARCHAR ,
endereco VARCHAR ,
    telefone INT ,
e-mail VARCHAR ,
PRIMARY KEY(idsocio));
CREATE TABLE editora (
ideditorSERIAL NOT NULL ,
nome VARCHAR ,
```

```
PRIMARY KEY(ideditor));
CREATE TABLE autor (
idautorSERIAL NOT NULL ,
nome VARCHAR ,
PRIMARY KEY(idautor));
CREATE TABLE Titulos (
isbnSERIAL NOT NULL ,
editora_ideditor INTEGER NOT NULL ,
titulo INTEGER ,
PRIMARY KEY(isbn),
    FOREIGN KEY(editora_ideditor) REFERENCES editora(ideditor));
CREATE TABLE videos (
Titulos_isbn INTEGER NOT NULL ,
duracao NUMERIC ,
PRIMARY KEY(Titulos_isbn),
    FOREIGN KEY(Titulos_isbn) REFERENCES Titulos(isbn));
CREATE TABLE exemplar (
idlivroSERIAL NOT NULL ,
Titulos_isbn INTEGER NOT NULL ,
data_entrada DATE ,
edicao INTEGER ,
PRIMARY KEY(idlivro),
    FOREIGN KEY(Titulos_isbn) REFERENCES Titulos(isbn));
CREATE TABLE livro (
Titulos_isbn INTEGER NOT NULL ,
paginas INTEGER ,
    peso NUMERIC ,
PRIMARY KEY(Titulos_isbn),
    FOREIGN KEY(Titulos_isbn) REFERENCES Titulos(isbn));
CREATE TABLE emprestimo (
idemprestimoSERIAL NOT NULL ,
socio_idsocio INTEGER NOT NULL ,
exemplar_idlivro INTEGER NOT NULL ,
data_emp DATE ,
data_devolucao DATE ,
    multa NUMERIC(7,2)) ,
data_reserva DATE ,
PRIMARY KEY(idemprestimo),
    FOREIGN KEY(exemplar_idlivro) REFERENCES exemplar(idlivro),
    FOREIGN KEY(socio_idsocio) REFERENCES socio(idsocio));
CREATE TABLE produz (
Titulos_isbn INTEGER NOT NULL ,
autor_idautor INTEGER NOT NULL ,
PRIMARY KEY(Titulos_isbn, autor_idautor),
    FOREIGN KEY(Titulos_isbn) REFERENCESTitulos(isbn),
    FOREIGN KEY(autor_idautor) REFERENCES autor(idautor));


CREATE TABLE cliente(idcliente SERIAL PRIMARY KEY, data JSONB);

INSERT INTO cliente (data) VALUES
(' { "nome" : "Ana" , "rua" : "Rua A" , "nr" : 1 , "complemento" : "101" , "bairro" : "Centro" , "cidade" :
"Rio" , "estado" : "RJ", "email" : "ana@mail.com" }'::JSONB) ,
(' { "nome" : "Rui", "rua" : "Rua B", "nr" : 2 , "complemento" : "102", "bairro" : "Centro" , "cidade" : "Rio" ,
"estado" : "RJ" , "email" : "ana@mail.com" }'::JSONB) ,
(' { "nome" : "Nei" , "rua" : "Rua C" , "nr" : 3 , "complemento" : "103" , "bairro" : "Centro" , "cidade" :
"Rio" , "estado" : "RJ" , "email" : "ana@mail.com" }'::JSONB)
```

Para verificar a criação correta executaremos um comando select simples:

```
SELECT * FROM cliente;
```

A seguir é apresentado o código para a criação da tabela locação e a inclusão de registros representando as locações realizadas por um cliente:

```
CREATE TABLE locacao(
nrlocacao SERIAL PRIMARY KEY,
data JSONB,
cliente_idcliente INTEGER REFERENCES cliente);

INSERT INTO locacao (data,cliente_idcliente) VALUES
('{"data_locacao":"2019-06-06","data_devolucao":"2019-06-
08","preco":"20.00","categoria":"romance","titulo":"Titulo A"}'::JSONB,1), --cast
('{"data_locacao":"2019-06-07","data_devolucao":"2019-06-
09","preco":"10.00","categoria":"humor","titulo":"Titulo B"}'::JSONB,1),
('{"data_locacao":"2019-06-08","data_devolucao":"2019-06-
10","preco":"15.00","categoria":"aventura","titulo":"Titulo C"}'::JSONB,2),
('{"data_locacao":"2019-06-09","data_devolucao":"2019-06-
11","preco":"17.00","categoria":"ficção","titulo":"Titulo D"}'::JSONB,2),
('{"data_locacao":"2019-06-10","data_devolucao":"2019-06-
12","preco":"20.00","categoria":"humor","titulo":"Titulo E"}'::JSONB,2);
```

Pode-se verificar o conteúdo do atributo JSONB com estecomando:

```
select data->>'titulo' fromlocacao
```

Para essa implementação, foi usado o recurso denominado CTE (common table expressions). Uma expressão de tabela comum é um conjunto de resultados temporários que pode ser referenciado em qualquer instrução SQL como as operações de manipulação de dados, incluindo SELECT, INSERT, UPDATE ou DELETE.

Com esse recurso é possível obter as seguintes vantagens:

1

A melhoria da legibilidade de consultas complexas — CTEs ajudam a organizar consultas complexas de maneira mais organizada e legível.

2

A criação de consultas recursivas que são aquelas que se referem as próprias consultas realizadas anteriormente; podendo ser úteis na realização de consultas em dados organizados de forma hierárquica ou em árvore.

Para retornar o conjunto de locações como um conjunto de documentos da coleção de locações de um cliente, usaremos o código a seguirque, além do CTE, usa algumas funções JSONB. São elas a função jsonb_pretty(), que retorna um campo JSON indentado JSON em texto, e a função jsonb_build_object(), que cria um objeto JSON a partir de uma lista de argumentos variados. Por convenção, a lista de argumentos consiste em chaves e valores alternados.

Pode-se verificar o conteúdo do atributo JSONB com este comando:

```
--cria a tabela temporária
WITH cliente_locacao AS
--agrupa os livros por autor
(
SELECT c.dataAS cliente,json_agg(l.data) as locacao
FROM cliente c LEFT JOIN locacao l
ON c.idcliente = l.cliente_idcliente
WHERE l.cliente_idcliente is not NULL
GROUP BY 1)

--aplicação da função jsonb_pretty() sobre a tabela temporária para apresentar os documentos indentados:
SELECT jsonb_pretty(cliente || jsonb_build_object('locacao', locacao))AS locacao
FROM cliente_locacao;
```

O resultado pode ser visto a seguir:


```
{
    "nr": 1,
    "rua": "Rua A",
    "nome": "Ana",
    "email": "ana@mail.com",
    "bairro": "Centro",
    "cidade": "Rio",
    "estado": "RJ",
    "locacao": [
       {
          "preco": "20.00",
          "titulo": "Titulo A",
          "categoria": "romance",
          "data_locacao": "2019-06-06",
          "data_devolucao": "2019-06-08"
       },
       {
          "preco": "10.00",
          "titulo": "Titulo B",
          "categoria": "humor",
          "data_locacao": "2019-06-07",
          "data_devolucao": "2019-06-09"
       }
    ],
    "complemento": "101"
}
{
    "nr": 2,
```

```
"rua": "Rua B",
"nome": "Rui",
"email": "ana@mail.com",
"bairro": "Centro",
"cidade": "Rio",
"estado": "RJ",
"locacao": [
   {
      "preco": "15.00",
      "titulo": "Titulo C",
      "categoria": "aventura",
      "data_locacao": "2019-06-08",
      "data_devolucao": "2019-06-10"
   },
   {
      "preco": "17.00",
      "titulo": "Titulo D",
      "categoria": "ficção",
      "data_locacao": "2019-06-09",
      "data_devolucao": "2019-06-11"
   },
   {
      "preco": "20.00",
      "titulo": "Titulo E",
      "categoria": "humor",
      "data_locacao": "2019-06-10",
      "data_devolucao": "2019-06-12"
   }
],
"complemento": "102"
```


Atividade

1. Assinale a afirmativa **errada** quanto ao entendimento do termo “agregado”em um projeto de banco de dados orientado a documentos:

a) Foi criado no surgimento do projeto orientado a domínios.
b) O agregado é um conjunto de objetos relacionados.
c) Facilita ainda o trabalho da implementação de *software* por parte dos desenvolvedores.
d) Na modelagem com agregados, os dados são definidos em unidades.
e) Seu uso é exclusivo para o modelo de dados relacional.

2. Assinale a afirmativa que indica um fator importante no projeto de um banco de dados orientado a documentos com a definição de agregados:

a) O número de agregados deve ser o mesmo da quantidade de tabelas existentes em um mesmo projeto no modelo relacional.
b) O máximo de agregações permitido é de três agregações.
c) Os agregados são criados para facilitar a execução das consultas.
d) O número de coleções é sempre superior ao número de tabelas de um projeto relacional.
e) O número de coleções é sempre igual ao número de tabelas de um projeto relacional.

---

3. Para que se possa trabalhar com o PostgreSQL no modelo orientado a documentos é necessário organizaros dados em coleções que são apresentadas como tabelas no relacional e tem em sua estrutura atributos do tipo JSONB. Assinale a alternativa **errada** quanto ao que faz este código:

```
WITH cliente_locacao AS
(
SELECT c.dataAS cliente, json_agg(l.data) as locacao
FROM cliente c LEFT JOIN locacao l
ON c.idcliente = l.cliente_idcliente
WHERE l.cliente_idcliente is not NULL
GROUP BY 1)

SELECT jsonb_pretty(cliente || jsonb_build_object('locacao', locacao)) AS locacao
FROM cliente_locacao ;
```

a) A primeira parte do código organiza os livros por autor.
b) A segunda parte do código aplicação da função jsonb_pretty() sobre a tabela temporária para apresentar os documentos indentados.
c) A primeira parte do código faz uso do recurso CTE – common tableexpressions.
d) São duas consultas totalmente independentes e seu resultado somente poderia ser alcançado com o uso de um operador *unionentre* as duas.
e) Foram usadas três funções para serem aplicadas em campos do tipo JSONB.

Notas

Referências

ELMASRI, R.; NAVATHE, S.B. **Sistemas de banco de dados**. 6. ed. São Paulo: Pearson Education do Brasil, 2018.

SADALAGE, P. J.;FOWLER, M. **NoSQLessencial–um guia conciso para o mundo emergente da persistência poliglota**. 1. ed. São Paulo: Novatec, 2013.

Próxima aula

- Características do banco de dados baseados em colunas;

- Vantagens do banco de dados colunar em relação ao modelo relacional;

- Comandos NOSQL para a definição da estrutura do banco de dados.

Explore mais

Para aprofundar seus conhecimentos, fica a sugestão de leitura do livro *Sistema de banco de dados* de Elmasri, R; Navathe, S.B.; Sistemas de Banco de Dados no do Cap. 24.3: Sistemas NOSQL baseados em documentos e MongoDB.

De modo e entender com mais detalhes do modelo orientado a documentos, faça a leitura do Capítulo 9 do livro Pramod J. Sadalage, Martim Fowler. NoSQL Essencial - Um Guia Conciso para o Mundo Emergente da Persistência Poliglota. 1. São Paulo: Novatec, 2013.