Banco de Dados NoSQL

Aula 4: Modelo de dados NoSQL baseado em chave-valor – parte II

Apresentação

Na Aula 3, iniciamos o estudo do modelo de banco de dados NoSQL chave-valor, onde verificamos as características principais do modelo, analisamos o conceito de *schemaless* e estudamos comandos para a definição da estrutura do banco de dados. Estudaremos agora, na Aula 4, os comandos necessários para a manipulação de dados nesse modelo. Conheceremos em mais detalhes o uso do formato JSON e o JSONB. Bom estudo!

Objetivo

- Escrever comandos NoSQL para a definição da estrutura de tabelas chave-valor com JSON;
- Escrever comandos NoSQL para a definição da estrutura de tabelas chave-valor com JSONB;
- Escrever comandos NoSQL para realizar a manipulação de dados em tabelas chave-valor.

Introdução

Na Aula 3, iniciamos o estudo do modelo chave-valor com o hstore no PostgreSQL e foram realizadas algumas consultas com o JSON[1] .

O formato é muito usado para troca de dados entre clientes e servidores web, pois seu antecessor, a linguagem de marcação extensível (XML), quando usado para esse fim, apresenta dificuldade no uso com a linguagem de programação interpretada JavaScript. Dessa forma, os desenvolvedores de aplicações para a web começaram também a preferir o uso do JSON em relação à XML (ELMASRI, 2018). Um dos motivos é que o JSON pode ser usado sem necessitar de um schema (como é o caso da XML), para a disponibilização de pares de valores-chave e listas ordenadas. Hoje, o JSON é o principal formato usado em servidores web. Apesar de criado a partir do JavaScript, ele é suportado nativamente ou por meio de bibliotecas por grande parte das principais linguagens de programação.

Além das vantagens do JSON, seu uso em bancos de dados acrescenta ao conjunto de funcionalidades normalmente presentes nos sistemas gerenciadores de bancos de dados, e permite a manipulação dos dados armazenados em JSON com os comandos da SQL. Dessa forma, uma consulta pode ser executada e exportada em JSON, facilitando o desenvolvimento de aplicações. Isso resultou no suporte nativo ao JSON em bancos de dados como o PostgreSQL e o MySQL.

A presente aula possui três objetivos e, por questões didáticas, o material será dividido em duas grandes partes:

Primeira parte

Apresenta os comandos para a definição da estrutura de tabelas em JSON e, logo em seguida, os comandos de manipulação nesse formato.

Segunda parte

Contém os comandos para a definição da estrutura em JSONB e os seus respectivos comandos para a manipulação dos dados.

Comentário

Foi considerada a divisão dessa forma para que o objetivo três não tivesse o conjunto de comandos de manipulação nos dois formatos em uma mesma seção do texto.

Comandos NoSQL para a definição da estrutura de tabelas chave-valor com JSON

Vamos iniciar o estudo do JSON com a definição da estrutura da tabela “pedidos” que será usada nos exemplos, tanto em relação à linguagem de definição de dados (DDL) quanto em relação à linguagem de manipulação de dados (DML). De forma semelhante ao comando usado na aula anterior com o hstore para a criação de tabela, também não é necessário definir a estrutura de um atributo usando o tipo de dado JSON.

A tabela tem apenas dois atributos iniciais, um identificador único com numeração sequencial automática e um atributo que armazenará os detalhes dos pedidos realizados por clientes.Nesse momento, não estamos discutindo as questões inerentes ao projeto de banco de dados NoSQL, o que será realizado posteriormente com o estudo de agregados. Vamos ao código para a criação da tabela com o atributo JSON:

```
CREATE TABLE pedidos (
      nrped serial PRIMARY KEY ,
      detalhes json NOT NULL ) ;
```

Logo após pode-se incluir registros:

```
INSERT INTO pedidos (detalhes) VALUES
( '{"cliente": "Nei", "produto": "Produto_1", "qtd": 8, "peso": "1 kg"}' ) ,
( '{"cliente": "Rui", "produto": "Produto_2", "qtd": 9, "peso": "10 kg"}' ) ,
( '{"cliente": "Lia", "produto": "Produto_3", "qtd": 15, "peso": "200 gr"}' ) ,
( '{"cliente": "Lia", "produto": "Produto_4", "qtd": 25, "volume": 2, "cor": "azul"}' ) ,
( '{"cliente": "Ana", "produto": "Produto_5", "qtd": 35, "volume": 3, "cor": "verde"}' ) ,
( '{"cliente": "Lia", "produto": "Produto_6", "qtd": 45, "volume": 4, "cor": "branco"}' ) ,
( '{"cliente": "Rui", "produto":"Produto_7", "qtd":55, "comprimento": "30 cm"}' ) ,
( '{"cliente": "Lia", "produto":"Produto_8", "qtd":65, "comprimento": "4 m"}' ) ,
( '{"cliente": "Rui", "produto":"Produto_9", "qtd":75, "comprimento": "50 mm"}' ) ,
( '{"cliente": "Eli", "produto": "Produto_10", "qtd": 85, "caixa_com": 20}' ) ,
( '{"cliente": "Eli", "produto": "Produto_11", "qtd": 95, "caixa_com": 30}' ) ,
( '{"cliente": "Ana", "produto": "Produto_12", "qtd": 105, "caixa_com": 60}' ) ;
```

Para verificar o resultado das inclusões e executar o comando SELECT abaixo, você deve ter observado que os registros de nrped 4 a 6 têm uma chave a mais (cor) que os demais.

```
SELECT * FROM pedidos ;
```

Figura 1 - Listagem dos registros com o atributo detalhes do tipo JSON

	nrped	detalhes
1	1	{"cliente": "Nei", "produto": "Produto_1", "qtd": 8, "peso": "1 kg"}
2	2	{"cliente": "Rui", "produto": "Produto_2", "qtd": 9, "peso": "10 kg"}
3	3	{"cliente": "Lia", "produto": "Produto_3", "qtd": 15, "peso": "200 gr"}
4	4	{"cliente": "Lia", "produto": "Produto_4", "qtd": 25, "volume": 2, "cor": "azul"}
5	5	{"cliente": "Ana", "produto": "Produto_5", "qtd": 35, "volume": 3, "cor": "verde"}
6	6	{"cliente": "Lia", "produto": "Produto_6", "qtd": 45, "volume": 4, "cor": "branco"}
7	7	{"cliente": "Rui", "produto":"Produto_7", "qtd":55, "comprimento": "30 cm"}
8	8	{"cliente": "Lia", "produto":"Produto_8", "qtd":65, "comprimento": "4 m"}
9	9	{"cliente": "Rui", "produto":"Produto_9", "qtd":75, "comprimento": "50 mm"}
10	10	{"cliente": "Eli", "produto": "Produto_10", "qtd": 85, "caixa_com": 20}
11	11	{"cliente": "Eli", "produto": "Produto_11", "qtd": 95, "caixa_com": 30}
12	12	{"cliente": "Ana", "produto": "Produto_12", "qtd": 105, "caixa_com": 60}

Fonte: JSON.

Para listar uma parte de uma chave como tipo JSON usa-se o operador "->". Nesse caso,serão listados três nomes de clientes;

```
SELECT detalhes -> 'cliente' AS cliente
FROM pedidos limit 3;
```

Figura 2 - Listagem de um valor de uma chave no formato JSON

	cliente
1	"Nei"
2	"Rui"
3	"Lia"

Fonte: JSON.

Uma observação é que não se pode usar o distinct e order by sobre o campo JSON. Assim, necessita-se transformar para texto com o operador " ->> ".

```
SELECT distinct detalhes ->> 'cliente' AS cliente
FROM pedidosorder by 1 ;
```

Figura 3 - Listagem de um valor de uma chave JSON como texto, sem repetição e com ordenação

	cliente
1	Ana
2	Eli
3	Lia
4	Nei
5	Rui

Fonte: JSON.

Esse operador também pode ser usado para gerar a listagem apresentando os pedidos feitos pela cliente de nome igual a 'Ana'. Vejamos o código para a consulta e o resultado apresentado na Figura 4.

```
SELECT * FROM pedidos
WHERE ( detalhes ->> 'cliente'::text ) = 'Ana' ;
```

Figura 4 - Resultado de consulta com a condição aplicada em um campo JSON

	nrped	detalhes
1	5	{"cliente": "Ana", "produto": "Produto_5", "qtd": 35, "volume": 3, "cor": "verde"}
2	12	{"cliente": "Ana", "produto": "Produto_12", "qtd": 105, "caixa_com": 60}

Fonte: JSON.

Caso a necessidade seja a de listar cada chave e o seu respectivo valor em separado.

```
SELECT json_each ( detalhes )FROM pedidos
WHERE ( detalhes ->> 'cliente'::text ) = 'Nei' ;
```

Figura 5 - Listagem de pares chave-valor de um registro de forma separada

	json_each
1	(cliente,"""Nei""")
2	(produto,"""Produto_1""")
3	(qtd,8)
4	(peso,"""1 kg""")

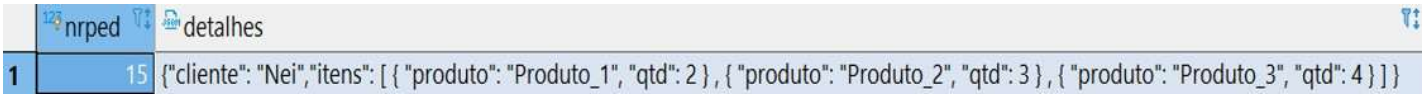
Fonte: JSON.

Agora vamos verificar como incluir itens em um pedido usando vetores (arrays) em JSON. Veja o exemplo apresentado a seguir incluindo três itens em um mesmo pedido. O resultado é apresentado na Figura 6.

```
INSERT INTO pedidos(detalhes) VALUES
( '{"cliente": "Nei","itens": [
      { "produto": "Produto_1", "qtd": 2 } ,
      { "produto": "Produto_2", "qtd": 3 } ,
      { "produto": "Produto_3", "qtd": 4 }] }') ;

SELECT * FROM pedidos where nrped = 13;
```

Figura 6 - Listagem de um registro que contém um array indicando itens de um mesmo pedido



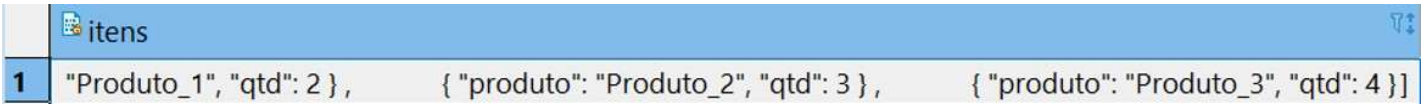
	nrped	detalhes
1	15	{"cliente": "Nei","itens": [{ "produto": "Produto_1", "qtd": 2 } , { "produto": "Produto_2", "qtd": 3 } , { "produto": "Produto_3", "qtd": 4 }] }

Fonte: JSON.

Para verificar somente os itens pode-se fazer assim (ver Figura 7):

```
SELECT detalhes -> 'itens' as itensFROM pedidos where nrped = 13;
```

Figura 7 - Listagem dos itens presentes na chave detalhes



	itens
1	"Produto_1", "qtd": 2 } , { "produto": "Produto_2", "qtd": 3 } , { "produto": "Produto_3", "qtd": 4 }]

Fonte: JSON.

Execute as consultas a seguir e verifique que é possível acessar cada elemento do array por meio de seu índice, que tem o valor iniciado em zero:

```
SELECT detalhes -> 'itens'->>0 as itens
FROM pedidos where nrped = 13;

SELECT detalhes -> 'itens'->>1 as itens
FROM pedidos where nrped = 13;
```

A consulta a seguir lista os tipos de dados usados na definição da estrutura da tabela. Esse tipo de consulta permite confirmar os tipos de dados usados e dessa forma avaliar o operador JSON adequado às respostas necessárias. Vejamos:

```
select distinct json_typeof(detalhes) as detalhes,
json_typeof(detalhes->'itens') as itens
from pedidos
```

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Figura 8 - Apresentação dos tipos de dados usados em um campo JSON e em um campo array

	detalhes	itens
1	object	[NULL]
2	object	array

Fonte: JSON.

Comandos NoSQL para a definição da estrutura de tabelas chave-valor com JSONB

O modelo chave-valor pode ser explorado ainda usando o formato JSONB, que está presente no PostgreSQL desde sua versão 9.4. Com ele, é possível realizar buscas mais rápidas e simples. De uma forma bem explícita, a maior diferença entre os tipos JSON e JSONB está justamente na performance:

O JSONB é muito mais rápido para fazer pesquisas, já que você pode indexar o conteúdo das chaves. Em termos de velocidade de escrita, os tempos são bem próximos, mas o JSON é um pouco mais rápido, sendo que isso acontece porque o JSONB precisa transformar os dados em uma estrutura nativa do PostgreSQL.

Vamos exercitar o modelo chave-valor com o tipo de dado JSONB. A seguir será apresentado o código para a criação da tabela "filmes".

```
CREATE TABLE filmes ( idfilme serial NOT NULL,dados jsonb);
```

A seguir é apresentado o código para a inclusão de registros de filmes contendo as respectivas classificações de gêneros em um atributo array em JSONB:

```
INSERT INTO filmes (dados) VALUES
( '{ "titulo": "Filme_A", "generos": ["Curta", "Romance", "Terror"], "publicado": false }' ) ,
( '{ "titulo": "Filme_B", "generos": ["Marketing", "Auto-ajuda", "Psicologia"], "publicado": true }' ) ,
( '{ "titulo": "Filme_C", "generos": ["Justiça", "Política"], "autores": ["Ana","Nei"],"publicado" : true
}' ) ,
( '{ "titulo": "Filme_D", "generos": ["Produtividade", "Tecnologia"]," publicado": true }' ) ,
( '{ "titulo": "Filme_E", "generos": ["Ficção", "Infanto-juvenil"], "publicado": true }' ) ;
```

A seguir, são apresentados os comandos para a manipulação dos dados em JSONB.

Para listar todos os registros da tabela, basta executar um comando simples. Observe que apenas um registro tem os autores indicados:

```
SELECT * FROM filmes ;
```

Figura 9 - Listagem com os livros cadastrados com destaque para campo JSONB

	idlivro	dados
1	2	{"titulo": "Filme_A", "generos": ["Curta", "Romance", "Terror"], "publicado": false}
2	3	{"titulo": "Filme_B", "generos": ["Marketing", "Auto-ajuda", "Psicologia"], "publicado": true}
3	4	{"titulo": "Filme_C", "autores": ["Ana", "Nei"], "generos": ["Justiça", "Política"], "publicado": true}
4	5	{"titulo": "Filme_D", "generos": ["Produtividade", "Tecnologia"], " publicado": true}
5	6	{"titulo": "Filme_E", "generos": ["Ficção", "Infanto-juvenil"], "publicado": true}

Fonte: JSONB.

As consultas a seguirlistam os títulos e os gêneros dos filmes em formato JSONB e em texto, respectivamente. A Figura 10 apresenta o resultado para a segunda consulta:

```
SELECT dados -> 'titulo' as titulo, dados -> 'generos' AS generos
FROM filmes ORDER BY 1 ;

SELECT dados ->> 'titulo' as titulo, dados ->> 'generos' AS generos
FROM filmes ORDER BY 1 ;
```

Figura 10 - Listagem com atributos JSONB em formato de texto

	titulo	generos
1	Filme_A	["Curta", "Romance", "Terror"]
2	Filme_B	["Marketing", "Auto-ajuda", "Psicologia"]
3	Filme_C	["Justiça", "Política"]
4	Filme_D	["Produtividade", "Tecnologia"]
5	Filme_E	["Ficção", "Infanto-juvenil"]

Fonte: JSONB.

Uma consulta para exibir somente registros que satisfazem a uma determinada condição em uma chave booleana.

```
SELECT * FROM filmes WHERE dados->'publicado' = 'false';
```

Com a função jsonb_array() pode-se listar cada elemento de um campo JSON em linhas separadas. Ver o resultado da consulta a seguir, na Figura 11.

```
SELECT jsonb_array_elements_text(dados -> 'generos') AS genero
FROM filmes WHERE idfilme = 3;
```

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Figura 11 - Listagem dos dados de uma chave em linhas separadas

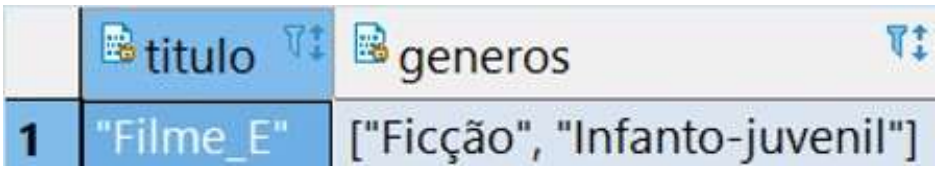
	genero
1	Justiça
2	Política

Fonte: JSONB.

Pode-se efetuar a busca na tabela a partir do valor de um elemento do array em JSONB. A Figura 12 apresenta o resultado da consulta.

```
SELECT dados->'titulo' as titulo,dados->'generos' as generos FROM filmes
WHERE dados->'generos' @> '["Ficção"]'::jsonb;
```

Figura 12 - Listagem de registro contendo gêneros com o valor de um item do array igual a Ficção



	titulo	generos
1	"Filme_E"	["Ficção", "Infanto-juvenil"]

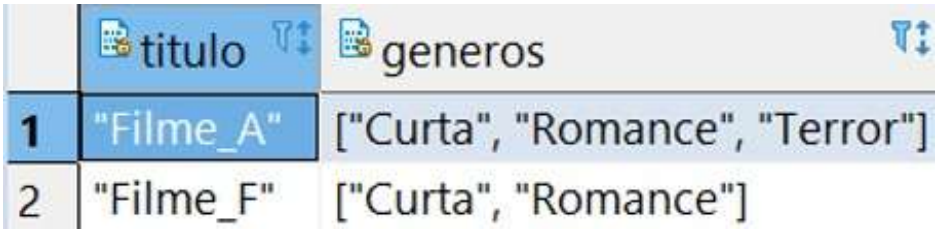
Fonte: JSONB.

Para executar a próxima consulta, será necessário incluir mais um registro para facilitar o entendimento da resposta apresentada.

```
INSERT INTO filmes (dados) VALUES
( '{ "titulo": "Filme_F", "generos": ["Curta", "Romance"], "publicado": false }' ) ;

SELECT dados -> 'titulo' as titulo, dados -> 'generos'AS generos
FROM filmes
WHERE dados->'generos' @> '["Curta", "Romance"]'::jsonb;
```

Figura 13 - Listagem dos registros que contêm os valores indicados na consulta no campo do tipo array



	titulo	generos
1	"Filme_A"	["Curta", "Romance", "Terror"]
2	"Filme_F"	["Curta", "Romance"]

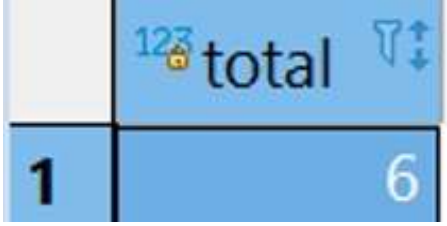
Fonte: JSONB.

Como o modelo chave-valor é schemaless, muitas vezes é necessário saber quais registros possuem determinadas chaves. As consultas a seguir apresentam os registros que contêm, respectivamente, o total de registros com as chaves‘autores’ e‘generos’. O resultado é o total igual a 1 para a primeira consulta e 6 para a segunda consulta (ver Figura 14).

```
SELECT COUNT(*) as total FROM filmes WHERE dados ? 'autores';

SELECT COUNT(*) as total FROM filmes WHERE dados ? 'generos';
```

Figura 14 - Resultado de consulta listando o total de registros que possuem a chave' generos' em um campo JSONB



Fonte: JSONB.

Por vezes, é necessária uma visualização “expandida” dos dados para melhor entendimento da estrutura dos tipos de dados em JSONB. Assim, na próxima consulta é apresentada a função jsonb_pretty( ) que possibilita a visão dos dados de forma indentada. Vejao resultado na Figura 15 apresentado no banco de dados e, logo a seguir, em forma textual (o que pode ser feito copiando o resultado da consulta e colando em um texto).

```
SELECT dados -> 'titulo' as titulo, dados -> 'generos'AS generos, jsonb_pretty( dados )

FROM filmes
order by 1 ;
```

Agora, depois de clicar no canto superior à esquerda do nome da coluna titulo e selecionar todo o resultado apresentado, podemos verificar o conteúdo de forma indentada; o que facilita o entendimento do conteúdo dos registros.

Figura 15 - Resultado da consulta com a função json_pretty() sem expansão do campo JSONB

	titulo	generos	jsonb_pretty
1	"Filme_A"	["Curta", "Romance", "Terror"]	{¶ "titulo": "Filme_A",¶ "generos": [¶ "Curta",¶ "Romance",¶ "Terror"¶],¶ "publicado": false¶}
2	"Filme_B"	["Marketing", "Auto-ajuda", "Psicologia"]	{¶ "titulo": "Filme_B",¶ "generos": [¶ "Marketing",¶ "Auto-ajuda",¶ "Psicologia"¶],¶ "publicado": true¶}
3	"Filme_C"	["Justiça", "Política"]	{¶ "titulo": "Filme_C",¶ "autores": [¶ "Ana",¶ "Nei"¶],¶ "generos": [¶ "Justiça",¶ "Política"¶],¶ "pub
4	"Filme_D"	["Produtividade", "Tecnologia"]	{¶ "titulo": "Filme_D",¶ "generos": [¶ "Produtividade",¶ "Tecnologia"¶],¶ " publicado": true¶}
5	"Filme_E"	["Ficção", "Infanto-juvenil"]	{¶ "titulo": "Filme_E",¶ "generos": [¶ "Ficção",¶ "Infanto-juvenil"¶],¶ "publicado": true¶}
6	"Filme_F"	["Curta", "Romance"]	{¶ "titulo": "Filme_F",¶ "generos": [¶ "Curta",¶ "Romance"¶],¶ "publicado": false¶}

Fonte: JSONB.

```
"Filme_A"     ["Curta", "Romance", "Terror "]     {
      "titulo": "Filme_A",
      "generos": [
            "Curta",
            "Romance",
            "Terror"
      ],
      "publicado": false
}
"Filme_B"     ["Marketing", "Auto-ajuda", "Psicologia"]     {
      "titulo": "Filme_B",
      "generos": [
            "Marketing",
            "Auto-ajuda",
            "Psicologia"
      ],
      "publicado": true
```

```
}
"Filme_C"     ["Justiça", "Política"]     {
       "titulo": "Filme_C",
       "autores": [
             "Ana",
             "Nei"
       ],
       "generos": [
             "Justiça",
             "Política"
       ],
       "publicado": true
}
"Filme_D"     ["Produtividade", "Tecnologia"]     {
       "titulo": "Filme_D",
       "generos": [
             "Produtividade",
             "Tecnologia"
       ],
       " publicado": true
}
"Filme_E"     ["Ficção", "Infanto-juvenil"]     {
       "titulo": "Filme_E",
       "generos": [
             "Ficção",
             "Infanto-juvenil"
       ],
       "publicado": true
}
"Filme_F"     ["Curta", "Romance"]     {
       "titulo": "Filme_F",
       "generos": [
             "Curta",
             "Romance"
       ],
       "publicado": false
}
```

Caso seja necessário adicionar uma chave, pode-se realizar a concatenação da seguinte forma:

```
       UPDATE filmes
       SET dados = dados || '{"ano":"2015"}'::jsonb
       WHERE idfilme = 3 ;
```

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Figura 16 - Resultado do acréscimo de uma chave no campo JSONB

	idfilme	dados
1	1	{"titulo": "Filme_A", "generos": ["Curta", "Romance", "Terror"], "publicado": false}
2	2	{"titulo": "Filme_B", "generos": ["Marketing", "Auto-ajuda", "Psicologia"], "publicado": true}
3	4	{"titulo": "Filme_D", "generos": ["Produtividade", "Tecnologia"], " publicado": true}
4	5	{"titulo": "Filme_E", "generos": ["Ficção", "Infanto-juvenil"], "publicado": true}
5	6	{"titulo": "Filme_F", "generos": ["Curta", "Romance"], "publicado": false}
6	3	{"ano": "2015", "titulo": "Filme_C", "autores": ["Ana", "Nei"], "generos": ["Justiça", "Política"], "publicado": true}

Fonte: JSONB.

Uma das vantagens de usar o formato JSONB do PostgreSQL em relação ao formato JSON é a possibilidade de usar índices GIN. Tais índices podem ser usados para pesquisar chaves ou pares de chave-valor com eficiência.

```
CREATE INDEX idx_publicados ON filmes USING GIN ((dados ->'publicado ’
```

Testes de desempenho de consultas usando ou não um índice só podem ser comparados com tabelas grandes, com milhões de linhas, daí a necessidade de uma carga controlada para realização de tais testes, o que está fora do escopo desta aula.

Atividade

1. De acordo com os comandos para a criação da tabela empréstimo e da inclusão de registros apresentados a seguir, indique o comando que pode ser usado para exibir os registros com o atributo JSONB de forma indentada:

```
CREATE TABLE emprestimo (idemprestimo serial PRIMARY KEY,
      emprestimo_detalhes jsonb NOT NULL);

INSERT INTO emprestimo (emprestimo_detalhes) VALUES
('{"emprestimo_detalhes":
      { "socio_id":1,
"nome":"Ana",
"endereco":"Rua A",
"telefone":"2222-4444",
"email": ana@eu.br ,
"emprestimo":
            [
                  {
"acervo_idlivro":100,
                  "data_emp":"2020-03-01",
"data_devolucao":"2020-03-11",
"multa":5.50,
                  "data_reserva":"2020-02-25"
            },
            {
            "exemplar_idlivro":101,
               "data_emp":"2020-03-01",
"data_devolucao":"2020-03-11"
}]} }');
```

a) SELECT jsonb_pretty(emprestimo_detalhes) FROM emprestimo.
b) SELECT jsonb_array_elements_text(emprestimo_detalhes) FROM emprestimo.
c) SELECT json_each (emprestimo_detalhes) FROM emprestimo.
d) SELECT json_typeof(emprestimo_detalhes) FROM emprestimo.
e) SELECT jsonb_json (emprestimo_detalhes) FROM emprestimo.

---

2. Com base na tabela apresentada a seguir e no registro a ser incluído nessa tabela, assinale a afirmativa que apresente os nomes dos autores em formato de texto:

```
CREATE TABLE acervo (
      idtitulo serial PRIMARY KEY,
      acervo_detalhes jsonb NOT NULL ) ;

INSERT INTO acervo ( acervo_detalhes ) VALUES
( '{ "isbn_id" : "978-85-7522-338-3 ",
      "titulo" : "NoSQL Essencial" ,
      "paginas" : 220 ,
      "peso" : 200 ,
      "editora" : "Pearson",
      "autor" : ["Pramod J. Sadalage" , "Martin Fowler"] ,
      "idlivro" : [100, 101, 102, 103]
} ') ;
```

a) select jsonb_array_elements_text (acervo_detalhes ->>'autor') as autor from acervo.
b) select jsonb_pretty (acervo_detalhes ->'autor') as autor from acervo.
c) select (acervo_detalhes_'autor')::JSONB as autor from acervo.
d) select acervo_detalhes -> 'autor' as autor from acervo.
e) select acervo_detalhes ->> 'autor' as autor from acervo.

---

3. Dados os registros apresentados na tabela a seguir. Considere o uso de um campo identificador e, para os campos que armazenarãoos demais detalhes dos pedidos, deve-se criar os campos: nome (texto), produto (texto), quantidade (inteiro), e o peso (texto). Utilize um array como tipo de dado JSONB em sua resposta incluindo todos os campos com exceção do identificador e do nome do cliente nesse tipo de dado. Assinale a afirmativa que contém o código para a inclusão de registros da tabela apresentada na questão contendo os dados de pedidos de compras de produtos.

nr	cliente	produto	quantidade	peso
1	Nei	Smartphone Xiaomi	8	
2		Tablet Samsung	9	
3		Smartphone iPhone	5	194 gramas

Atenção! Para visualização completa da tabela utilize a rolagem horizontal

a) INSERT INTO pedidos(detalhes) VALUES (' {'cliente' : 'Nei', "itens": ['produto':'Smartphone Xiaomi', 'qtd':8}'),(
'{'cliente':'Rui','produto':'Tablet Samsung','qtd':9}'),('{'cliente': 'Lia', 'produto': 'Smartphone iPhone 6', 'qtd':5, 'peso':194 gramas }] } ') ;
b) INSERT INTO pedidos(detalhes) VALUES ('{"cliente"=>"Nei","itens": ["produto"=>"Smartphone Xiaomi","qtd"=>8}'),
('{"cliente"=>"Rui","produto"=>"Tablet Samsung","qtd"=>9}'),
('{"cliente"=>"Lia","produto"=>"Smartphone iPhone","qtd"=>5,"peso"=>194 gramas }] } ') ;
c) INSERT INTO pedidos(detalhes) VALUES
('{"cliente":"Nei","itens":[
{"produto":"Smartphone Xiaomi","qtd":8},{"produto":"Tablet Samsung","qtd":9},
{"produto":"Smartphone iPhone","qtd":5,"peso":194gramas }]}');
d) INSERT INTO pedidos(detalhes) VALUES ("cliente"=>"Nei","itens": ["produto"=>"Smartphone Xiaomi","qtd"=>8),
('cliente"=>"Rui","produto"=>"Tablet Samsung","qtd"=>9'),
('cliente"=>"Lia","produto"=>"Smartphone iPhone","qtd"=>5,"peso"=>194 gramas }]}') ;
e) INSERT INTO pedidos(detalhes) VALUES ('{"cliente":"Nei","itens": ["produto":"Smartphone Xiaomi","qtd":8}'),
('{"cliente":"Rui","produto":"Tablet Samsung","qtd":9}'),
('{"cliente":"Lia","produto":"Smartphone iPhone","qtd":5,"peso":194 gramas }]}') ;

Notas

JSON [1]

O JSON (JavaScript Object Notation) tem sido muito utilizado, desde seu lançamento em 2000, em aplicações que necessitam de publicação baseada em texto. Pode-se classificar o JSON como um formato genérico e que usa um número limitado de tipos de dados composto por números, strings, booleanos (true/false), vetores, objetos e nulo

Referências

ELMASRI, R.; NAVATHE, S.B.; **Sistemas de Banco de Dados**. 6ª. Edição; São Paulo: Pearson Education do Brasil; 2018 (BIBLIOTECA VIRTUAL) disponível em: https://plataforma.bvirtual.com.br/Acervo/Publicacao/1992

SADALAGE,J.Pramod;FOWLER,Martin. **NoSQL Essencial - Um Guia Conciso para o Mundo Emergente da Persistência Poliglota**. 1. São Paulo: Novatec, 2013.

Próxima aula

Na próxima aula, daremos continuidade aos estudos vistos aqui, abordando os seguintes conteúdos:

- Características do banco de dados orientado a documentos;
- Formatos de documentos permitidos para uso nesse modelo;
- Comandos NoSQL para a definição da estrutura do banco de dados orientado a documentos.

Explore mais

Para aprofundar seus conhecimentos fica a sugestão de leitura do livro *Sistema de banco de dados* de Elmasri e Navathe, no Cap. 24.4: Armazenamento chave-valor em NOSQL.

De modo a entender com mais detalhes o modelo chave-valor, faça a leitura do Capítulo 8 Bancos de dados de chave-valor do livro Sadalage e Fowler *NoSQL Essencial - Um Guia Conciso para o Mundo Emergente da Persistência Poliglota*.