Banco de Dados NoSQL

Aula 3: Modelo de dados NoSQL baseado em chave-valor – parte I

Apresentação

Na Aula 3, será estudado o modelo de banco de dados NoSQL chave-valor. Verificaremos as características desse modelo e analisaremos o conceito de schemaless.Estudaremos os comandos necessários para a criação das tabelas e verificaremos como usar campos em formatos JSON para o armazenamento chave-valor. Conheceremos também como trabalhar com a extensão HSTORE no PostgreSQL para manipular atributos chave-valor.

Objetivo

- Reconhecer as características do banco de dados baseado em chave-valor;
- Examinar o conceito de schemaless (sem esquema);
- Escrever comandos NoSQL para a definição da estrutura do banco de dados.

Introdução

Os bancos de dados que seguem o modelo chave-valor são os mais simples dos bancos de dados NoSQL, sendo por isso o modelo mais usado. Ele é conhecido pelo formato como armazena os dados, o qual utiliza um conjunto de chaves-valores. Portanto,são também conhecidos como bancos de dados de pares de chave-valor ou *key-value pair databases* (KVP). Os sistemas gerenciadores de bancos de dados (SGBD) desse modelo de dados contêm recursos para a gerência desses grupos de pares de chave-valor.

Exemplo

Como exemplos de banco de dados NoSQL do modelo chave-valor, podem ser citados o Redis, Amazon Dynamo DB, Microsoft Azure Cosmos DB, todos multimodelos, por admitirem outros modelos de dados; e outros chave-valor puros, como Memcached e etcd. Outro exemplo é a extensão HSTORE do PostgreSQL, que será usada nesta aula.

Os pares são compostos primeiro pela chave seguida do valor do dado. A chave funciona como um identificador para um item dos dados e o valor armazena o conteúdo de acordo com o tipo de dado. Pode-se comparar a chave como o nome do atributo ou coluna no modelo relacional. Um registro no modelo chave-valor (PRAMOD, 2013) é um atributo composto por diferentes atributos e que pode ser de diferentes tipos de dados também.

Características do banco de dados baseado em chave-valor

1

De modo a detalhar como os dados são armazenados, a chave pode ser usada de diversas formas, mas isso depende do SGBD, que pode impor algumas limitações. Por motivos de desempenho, deve-se evitar o uso de uma chave muito longa. A chave deve seguir uma padronização para permitir a administração, assegurar a qualidade dos dados e manter a consistência. Os valores armazenados também podem ser uma lista ou ainda um outro par chave-valor encapsulado em outro objeto.

2

Ainda em relação ao valor armazenado, dentro de um campo de uma tabela no banco de dados chave-valor, pode-se armazenar tipos de dados diferentes para as chaves. Cabe ao DBA verificar as funcionalidades do SGBD que utilizará na aplicação, pois alguns SGBDs que usam o modelo de armazenamento baseado em chaves permitem a definição do tipo de dados para o valor, por exemplo, pode-se determinar que o valor deve ser um atributo inteiro ou um texto. Porém, outros bancos de dados não oferecem essa funcionalidade e, nessa situação, o valor armazenado pode ser de qualquer tipo de dado.

3

Por conta de sua simplicidade, os SGBDs chave-valor são muito usados em aplicações que necessitam de velocidade nas pesquisas. Eles têm um desempenho superior aos relacionais, principalmente quando são usados em operações que necessitam de velocidade nas leituras e gravações de dados. Isso devido ao fato de que seu suporte ao processamento não segue o modelo clássico de transações usado nos bancos relacionais com base nas propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade) (ELMASRI, 2018). Ou seja, a velocidade foi o critério primordial para a indicação dos tipos de aplicações mais adequadas para seu uso, assim como suas formas de armazenamento de dados, com um suporte de transações diferente dos relacionais.

Assim, pode-se dizer que esse modelo de dados é muito eficaz quanto ao dimensionamento de acordo com as aplicações que operam com dados não transacionais de alta velocidade. Por não oferecer suporte transacional, exigem dos desenvolvedores das aplicações a codificação para o tratamento dos dados, a replicação e as rotinas de tolerância a falhas, tendo em vista que não são expressamente controlados pela própria tecnologia usada no modelo de dados. Por outro lado, essa escolha facilita o uso do modelo em aplicações web e dispositivos móveis, ou ainda em aplicações com a inclusão automática de dados.

Os bancos de dados no modelo chave-valor são bem mais fáceis de serem particionados e possibilitam a escalabilidade do tipo horizontal, algo que outros modelos de dados não oferecem. Assim, dependendo do volume dos

dados das aplicações, os administradores de bancos de dados (DBA), juntamente com a equipe responsável pela arquitetura da aplicação, podem sugerir o particionamento desses dados, assim como sua replicação com clusters para obter maior escalabilidade e disponibilidade, eliminando assim o perigo de ter um ponto único de falha no sistema e aumentando a performance.

Conceito de *schemaless* (sem esquema)

Durante o projeto de um banco de dados relacional, é necessário definir duas coisas:

1

A estrutura (campos ou atributos) das tabelas.

2

Os tipos de dados para cada um dos atributos.

Comentário

Dependendo da aplicação, alguns desses atributos nem sempre são usados e, em outra situação, a predefinição da estrutura da tabela com os seus atributos nem sempre é tão fácil de ser realizada, pois os dados podem variar muito quanto ao conjunto de atributos que efetivamente serão usados pelos usuários.

Para facilitar a inclusão de registros, o modelo chave-valor não exige um esquema predefinido como nos bancos de dados relacionais. Dessa forma, oferecem mais flexibilidade para atender às necessidades das aplicações.

Exemplo

Um exemplo de uso para essa funcionalidade seria incluir todas as postagens de usuários em blogs em sistemas de gerenciamento de conteúdo, apresentando grandes variações no conteúdo. Assim, podemos definir schemaless ou "sem esquema" como o banco de dados que não possui a estrutura de suas tabelas de forma fixa e preestabelecida.

De modo a manter um padrão de formato com amplo conhecimento, esse modelo permite o armazenamento de dados no formato JSON, também muito usado em intercâmbio de dados. Nele, pode-se alterar a estrutura de dados de acordo com a necessidade da aplicação.

Comandos NoSQL para a definição da estrutura do banco de dados

A seguir serão apresentados comandos para uso do modelo chave-valor, inicialmente, usando a extensão hstore do SGBD PostgreSQL.

Ela é uma contribuição (contrib) disponível desde a versão 8.4 (2006) do PostgreSQL. Sua proposta consiste em ser útil para armazenar conjuntos de dados compostos por chaves-valor e que são armazenados em uma única coluna (atributo) de uma tabela conforme a proposta *schemaless* do banco NoSQL. O tipo do dado usado no armazenamento é um atributo composto e pode utilizar tipos de dados diferentes.

O tipo de dados *hstore* é muito útil em aplicações que envolvem dados semiestruturados ou linhas com muitos atributos que nem sempre são consultados ou raramente usados. Pode-se inserir atributos, consultando-se depois com base nas suas chaves ou nos seus valores.

Para usar o *hstore*, há a necessidade de habilitar a extensão, para tal, basta executar o comando abaixo apenas uma vez. Com o PostgreSQL instalado (versão 8.4 ou superior), você habilitará o interpretador de comandos dessa extensão com o comando:

```
CREATE EXTENSION hstore;
```

Para fazer uso na criação de uma tabela, basta usar o *hstorecomo* os demais tipos de dados normalmente usados na definição da estrutura de uma tabela. Vejamos um exemplo para a criação de uma tabela cujo objetivo é armazenar registros de *livros*.

```
CREATE TABLE livros (
nr serial primary key ,
titulo varchar(255) ,
detalhes hstore );
```

O atributo *detalhes* na tabela foi criado com o *hstore* para armazenar, de forma composta, os atributos do livro, como ISBN (que é um número internacional de identificação de livros), peso e tipo de capa. Observe que não há uma especificação para cada tipo de dado em cada chave, assim, tem-se atributos textuais e, no caso do atributo *paginas*, esse é do tipo inteiro. Também não houve uma definição da quantidade de chaves, que de forma similar no modelo relacional, seriam as colunas.

Agora, será apresentado o código para a inclusão de registros na tabela *livros*. Observe a forma usada para indicar o valor para cada uma das chaves:

```
INSERT INTO livros ( titulo, detalhes ) VALUES
( 'Mastering PostgreSQL 10',
'
      "paginas" => 428 ,
      "assunto" => "PostgreSQL"
,
      "idioma" => "inglês" ,
      "ISBN" => "1788472292" ,
      "ISBN-13" => "978-1788472296 ",
      "peso" => "798 g"
' ) ;
```

Os dados que inserimos na coluna hstore são uma lista de pares chave => valor separados por vírgula. As chaves e os valores para os atributos textuais são citados usando aspas duplas (""). Agora, na inclusão do próximo registro na tabela, há uma diferença no conjunto de chaves presente no campo detalhes, que contém uma chave a menos do que o primeiro registro e, uma outra denominada publicador, não presente na primeira inclusão. Vejamos o comando insert para esse exemplo:

```
INSERT INTO livros ( titulo, detalhes )
VALUES ( 'NOSQL Essencial: Um Guia Conciso para o Mundo Emergente da Persistência
Poliglota',
'
      "paginas" => 216,
      "publicador" => "Novatec" ,
      "idioma" => "português" ,
      "ISBN" => "8575223380" ,
      "peso" => "381 g"
') ;
```

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Figura 1 - Exemplos de dois registros armazenados com o uso de atributo do tipo hstore

	nr	titulo	detalhes
1	1	Mastering PostgreSQL 10	"ISBN"=>"1788472292", "peso"=>"798 g", "idioma"=>"inglês", "ISBN-13"=>"978-1788472296 ", "assunto"=>"PostgreSQL", "paginas"=>"428"
2	2	NOSQL Essencial: Um Guia Concis	"ISBN"=>"8575223380", "peso"=>"381 g", "idioma"=>"português", "paginas"=>"216", "publicador"=>"Novatec"

Fonte: PostgreSQL.

A Figura 1 apresenta o resultado de uma consulta SELECT comum após a inclusão dos dois registros da tabela *livros*. Os dois primeiros atributos são atributos comuns, do tipo inteiro, e o segundo, do tipo texto; no atributo *detalhes*, do tipo hstore, podemos observar a diferença no conjunto de chaves no final dos registros.

Comentário

Observe ainda que o valor das páginas no INSERT não foi colocado entre aspas, ou seja, foi incluído com número inteiro, mas o hstore armazenou na chave *peso* como tipo de dado textual.

Execute o comando de seleção simples abaixo para obter o resultado apresentado na Figura 1:

```
select * from livros;
```

Pode-se atualizar o conteúdo de uma chave em um campo *hstore* com a aplicação da função REPLACE(). No código abaixo, a função REPLACE altera o conteúdo do atributo *detalhes*, substituindo o valor da chave *paginas*. Para isso, foi necessário utilizar a função de conversão de tipo de dados CAST para alterar o valor da chave para texto e depois retornar o valor para *hstore*. Vejamos o código deste update:

```
UPDATE livros
SET detalhes = REPLACE(detalhes::text,'"paginas"=>"428"'::text,'"paginas"=>"429"'::text)::hstore
WHERE detalhes ->'ISBN' = '1788472292';
```

Você pode visualizar o resultado com o comando de seleção abaixo:

```
SELECT * FROM livros
WHERE detalhes->'ISBN' = '1788472292';
```

Agora não vamos usar mais a extensão *hstore*, mas sim o tipo de dados JSON (*JavaScript Object Notation*) na criação de uma tabela.

O JSON é um formato-padrão aberto que consiste em pares de chaves-valor.

O principal uso do JSON é o intercâmbio de dados entre um servidor e uma aplicação na web.

De forma diferente da apresentada por outros formatos, o JSON é um formato em texto legível por humanos.

Vejamos um exemplo para a criação da tabela anterior para o armazenamento de livros, onde o campo *detalhes* é agora do tipo JSON.

Primeiro, exclua a tabela anterior que usava o campo *detalhes* em hstore com o comando:

```
DROP TABLE livros;
```

Vamos efetuar a criação da tabela com o tipo JSON. É bom destacar que não há a definição da estrutura do campo que usa esse tipo de dado; o atributo vai estar pronto para receber as chaves de acordo com a necessidade no momento do armazenamento dos dados.

Vejamos o código para a criação da tabela livros com o JSON:

```
CREATE TABLE livros (
nr serial primary key ,
titulo varchar(255) ,
detalhes json ) ;
```

Execute o comando a seguirpara a visualização da tabela *livros*:

```
SELECT * FROM livros ;
```

A seguir temos o INSERT na tabela livros, onde os pares chave-valor são delimitados por chaves “{ }”indicandoo bloco relativo ao atributo *detalhes*:

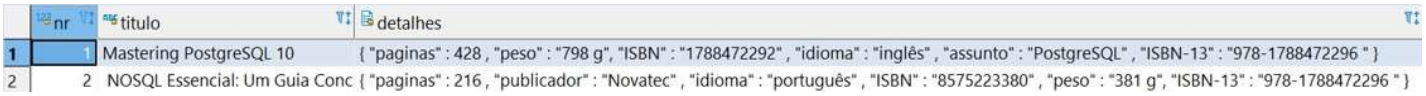
```
INSERT INTO livros ( titulo, detalhes ) VALUES
( 'Mastering PostgreSQL 10',
'{
      "paginas" : 428 ,
      "peso" : "798 g",
      "ISBN" : "1788472292" ,
      "idioma" : "inglês" ,
      "assunto" : "PostgreSQL" ,
      "ISBN-13" : "978-1788472296 "
}' ) ;
```

A seguir, é apresentado outro INSERT com diferença na quantidade de pares chave-valor. O resultado das inclusões é apresentado na Figura 2.

```
INSERT INTO livros ( titulo, detalhes )
VALUES ( ' NOSQL Essencial: Um Guia Conciso para o Mundo Emergente da Persistência Poliglota ',
```

```
'{
      "paginas" : 216 ,
      "publicador" : "Novatec" ,
      "idioma" : "português" ,
      "ISBN" : "8575223380" ,
      "peso" : "381 g",
      "ISBN-13" : "978-1788472296 "
}' ) ;
```

Figura 2 - Exemplos de dois registros com o uso de atributo do tipo JSON e com diferença na composição do atributo detalhes



	nr	titulo	detalhes
1	1	Mastering PostgreSQL 10	{ "paginas" : 428 , "peso" : "798 g", "ISBN" : "1788472292" , "idioma" : "inglês" , "assunto" : "PostgreSQL" , "ISBN-13" : "978-1788472296 " }
2	2	NOSQL Essencial: Um Guia Conc	{ "paginas" : 216 , "publicador" : "Novatec" , "idioma" : "português" , "ISBN" : "8575223380" , "peso" : "381 g", "ISBN-13" : "978-1788472296 " }

Fonte: PostgreSQL.

Para a atualização do valor de parte de uma chave também podemos usar a função REPLACE(). Abaixo segue um exemplo para atualizar o valor da chave *paginas* sendo informado o valor do ISBN como predicado de consulta: Aqui é valor mesmo, não deveria ser número?

```
UPDATE livros
SET detalhes = replace(detalhes::text,'"paginas" : 428 '::text,'"paginas" : 429 '::text)::json
WHERE detalhes ->> 'ISBN' = '1788472292'
```

A seguir iniciaremos a apresentação de consultas simples, começando com o atributo do tipo *hstore*. Note que, para executar essas consultas, será necessário remover a tabela livros (DROP TABLE) e recriá-la (CREATE TABLE) com o atributo *detalhes* do tipo *hstore*, como no exemplo anterior.

Para listar uma chave específica pertencente ao campo *detalhes* o resultado é exibido na Figura 3:

```
SELECT detalhes -> 'idioma' AS "Idioma"
FROM livros;
```

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Figura 3 - Resultado de uma consulta listando apenas uma chave de um atributo hstore

	Idioma
1	inglês
2	português

Fonte: PostgreSQL.

Consulta para selecionar um registro a partir de uma condição imposta a parte de uma chave específica. Ver Figura 4 a seguir:

```
SELECT *
FROM livros
WHERE detalhes->'ISBN' = '1788472292';
```

Figura 4 - Resultado de uma consulta listando os dados a partir de uma condição informada para apenas uma chave de um atributo hstore

	nr	titulo	detalhes
1	1	Mastering PostgreSQL 10	"ISBN"=>"1788472292", "peso"=>"798 g", "idioma"=>"inglês", "ISBN-13"=>"978-1788472296 ", "assunto"=>"PostgreSQL", "paginas"=>"428"

Fonte: PostgreSQL.

A consulta anterior também pode ser realizada a partir de uma busca em parte do valor de uma chave, como estecódigo:

```
SELECT *
FROM livros
WHERE detalhes -> 'ISBN' ILIKE '%88%' ;
```

A seguir é apresentada uma consulta para listar somente os nomes das chaves de forma distinta e o resultado é apresentado na Figura 5.

```
SELECT distinct akeys(detalhes)
FROM livros
order by 1;
```

Figura 5 - A consulta lista os nomes das chaves que realmente foram usadas em cada registro

	akeys
1	{ISBN,peso,idioma,ISBN-13,assunto,paginas}
2	{ISBN,peso,idioma,paginas,publicador}

Fonte: PostgreSQL.

Dependendo da quantidade de chaves presentes no atributo hstore, pode ser que a visualização se torne difícil. Assim, a consulta anterior pode ser executada de outra maneira; na forma de um conjunto. O resultado é apresentado a seguir, na Figura 6.

Figura 6 - Resultado da consulta que lista as chaves presentes no atributo hstore como um conjunto

	skeys
1	ISBN
2	peso
3	idioma
4	ISBN-13
5	assunto
6	paginas
7	ISBN
8	peso
9	idioma
10	paginas
11	publicador

Fonte: PostgreSQL.

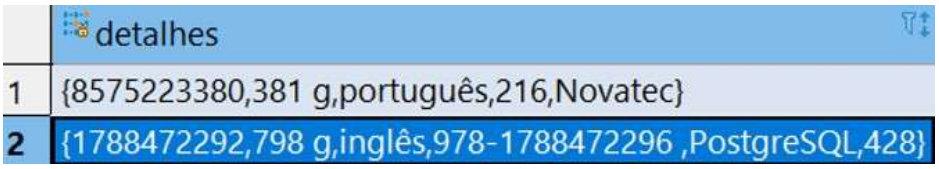
Na consulta apresentada na Figura 6, foi possível observar repetições nos valores das chaves. Lembrando que, para retirar as repetições, podemos acrescentar um distinct como está na consulta a seguir:

```
SELECT distinct skeys(detalhes)
FROM livros;
```

A seguir temos uma consulta para listar apenas os valores presentes no atributo hstore, o que pode ser visto na Figura 7.

```
SELECT distinct avals(detalhes) os detalhes
FROM livros;
```

Figura 7 - Listagem apenas dos valores apresentados no atributo hstore



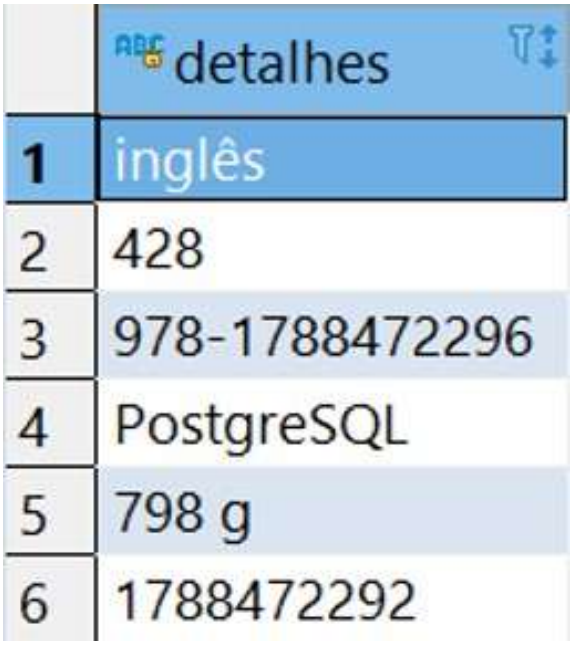
	detalhes
1	{8575223380,381 g,português,216,Novatec}
2	{1788472292,798 g,inglês,978-1788472296 ,PostgreSQL,428}

Fonte: PostgreSQL.

Em outra consulta para extrair os valores do atributo hstore na forma de um conjunto, o que pode ser visto na Figura 8:

```
SELECT distinct svals(detalhes) as detalhes
FROM livros
WHERE detalhes->'ISBN' = '1788472292';
```

Figura 8 - Consulta apresentando a listagem dos valores contidos no atributo hstore de forma separada



	detalhes
1	inglês
2	428
3	978-1788472296
4	PostgreSQL
5	798 g
6	1788472292

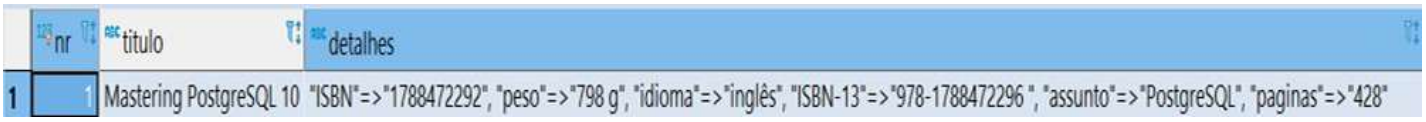
Fonte: PostgreSQL.

A seguir, é apresentada a consulta para gerar uma listagem com os registros que têm uma determinada chave na composição de um atributo hstore. No exemplo, foi listado apenas o registro que possui a chave ISBN-13. O resultado pode ser apresentado na Figura 9 a seguir:

```
SELECT *
```

```
FROM livros
WHERE detalhes ? 'ISBN-13' ;
```

Figura 9 - Listagem do registro que contém no atributo detalhes a chave igual a ISBN-13



	nr	titulo	detalhes
1	1	Mastering PostgreSQL 10	"ISBN"=>"1788472292", "peso"=>"798 g", "idioma"=>"inglês", "ISBN-13"=>"978-1788472296 ", "assunto"=>"PostgreSQL", "paginas"=>"428"

Fonte: PostgreSQL.

Uma outra forma de pesquisa por parte de uma chave, semelhante à consulta anterior, é apresentada a seguir. Nela, podemos pesquisar por um determinado predicado de consulta.

```
SELECT titulo
FROM livros
WHERE detalhes @>'"peso"=>"798 g"'::hstore ;
```

A tarefa de conversão de dados de um formato para outro é muito comum nas atividades de bancos de dados e, principalmente, em aplicações que possuem bancos de dados de diferentes modelos. A próxima consulta faz a conversão do formato hstore para o JSON. O resultado é apresentado na Figura 10.

```
SELECT hstore_to_json (detalhes) json
FROM livros ;
```

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Figura 10 - Conversão de tipo de dado hstore para JSON

	json
1	{"ISBN": "1788472292", "peso": "798 g", "idioma": "inglês", "ISBN-13": "978-1788472296 ", "assunto": "PostgreSQL", "paginas": "428"}
2	{"ISBN": "8575223380", "peso": "381 g", "idioma": "português", "paginas": "216", "publicador": "Novatec"}

Fonte: PostgreSQL.

No último exemplo da Aula 3,apresentaremos o código para efetuar a conversão de um atributo hstore para sets (conjuntos). O resultado é apresentado na Figura 11.

```
SELECT titulo,(EACH(detalhes) ).*
```

```
FROM livros;
```

Figura 11 - Conversão de tipo de dado hstore para JSON

	titulo	key	value
1	Mastering PostgreSQL 10	ISBN	1788472292
2	Mastering PostgreSQL 10	peso	798 g
3	Mastering PostgreSQL 10	idioma	inglês
4	Mastering PostgreSQL 10	ISBN-13	978-1788472296
5	Mastering PostgreSQL 10	assunto	PostgreSQL
6	Mastering PostgreSQL 10	paginas	428
7	NOSQL Essencial: Um Guia Conciso para o Mundo Emergente da Persistência Poliglota	ISBN	8575223380
8	NOSQL Essencial: Um Guia Conciso para o Mundo Emergente da Persistência Poliglota	peso	381 g
9	NOSQL Essencial: Um Guia Conciso para o Mundo Emergente da Persistência Poliglota	idioma	português
10	NOSQL Essencial: Um Guia Conciso para o Mundo Emergente da Persistência Poliglota	paginas	216
11	NOSQL Essencial: Um Guia Conciso para o Mundo Emergente da Persistência Poliglota	publicador	Novatec

Fonte: PostgreSQL.

Atividade

1. Assinale a afirmativa que indica um fator importante para o desempenho dos bancos de dadosNoSQL:

a) A característica de ser schemaless.
b) Não usar apenas a SQL.
c) A escalabilidade vertical.
d) A limitação de que os valores presentes em um conjunto de pares chave-valor sejam do mesmo tipo de dado.
e) Ter suporte de transações diferente do relacional.

2. Assinale a afirmativa correta quanto à criação de tabelas eàcaracterística schemaless no modelo chave-valor:

a) Exige a definição da estrutura da tabela.
b) Exige a definição dos tipos de dados utilizados.
c) Não exige a predefinição da estrutura da tabela.
d) Exige a definição da quantidade de tabelas da aplicação.
e) Não exige o uso de um schema, permitindo a criação das tabelas apenas no schema public.

3. Assinale a afirmativa que contém o código para a inclusão dos registros da tabela apresentada na questão para o armazenamento de produtos. Considere o uso de um campo identificador e campos necessários para os demais detalhes do produto. Utilize o tipo de dado JSON em sua resposta. Considere os registros apresentados na tabela a seguir:

nr	cliente	produto	Qtd	peso
1	Nei	Smartphone Xiaomi	8	
2	Rui	Tablet Samsung	9	

3	Lia	Smartphone iPhone	5	194 gramas

Atenção! Para visualização completa da tabela utilize a rolagem horizontal

a) INSERT INTO pedidos(detalhes) VALUES (‘{‘cliente’:’Nei’, ‘produto’: ‘Smartphone Xiaomi’,’qtd’:8}’),(‘{‘cliente’: ‘Rui’, ‘produto’: ‘Tablet Samsung’,’qtd’:9}’),(‘{‘cliente’:’Lia’,’produto’:’Smartphone iPhone’,’qtd’:5,’peso’:194}’);

b) INSERT INTO pedidos(detalhes) VALUES ('{"cliente"=>"Nei","produto"=>"Smartphone Xiaomi","qtd"=>8}'),
('{"cliente"=>"Rui","produto"=>"Tablet Samsung","qtd"=>9}'),('{"cliente"=>"Lia","produto"=>"Smartphone iPhone","qtd"=>5,"peso"=>194}');

c) INSERT INTO pedidos(detalhes) VALUES ('{"cliente"="Nei","produto"="Smartphone Xiaomi","qtd"=8}'),
('{"cliente"="Rui","produto"="Tablet Samsung","qtd"=9}'),
('{"cliente"="Lia","produto"="Smartphone iPhone","qtd"=5,"peso"=194}');

d) INSERT INTO pedidos(detalhes) VALUES ("cliente"=>"Nei","produto"=>"Smartphone Xiaomi","qtd"=>8),
('cliente"=>"Rui","produto"=>"Tablet Samsung","qtd"=>9'),
('cliente"=>"Lia","produto"=>"Smartphone iPhone","qtd"=>5,"peso"=>194');

e) INSERT INTO pedidos(detalhes) VALUES ('{"cliente":"Nei","produto":"Smartphone Xiaomi","qtd":8}'),('{"cliente":"Rui","produto":"Tablet Samsung","qtd":9}'),
('{"cliente":"Lia","produto":"Smartphone iPhone","qtd":5,"peso":194}');

Notas
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

Pramod J. Sadalage, Martim Fowler. 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:

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

Explore mais

Leia os textos:

- Refactoring
- Cyclomatic complexity density and software maintenance productivity