Estudo dirigido de Cassandra

# Keyspace

Em Apache Cassandra, o conceito de "Keyspace" é semelhante a um "banco de dados" em sistemas de gerenciamento de banco de dados relacional. Um keyspace é o espaço de nomes de nível mais alto que determina como os dados são replicados em nós.

Sintaxe

A sintaxe básica para criar um keyspace é:

CREATE KEYSPACE <keyspace\_name>

WITH REPLICATION = { 'class' : '<replication\_class>', '<replication\_option>' : '<option\_value>', ... };

Onde:

<keyspace\_name>: O nome que você deseja dar ao keyspace.

<replication\_class>: A estratégia de replicação que você deseja usar. As mais comuns são SimpleStrategy e NetworkTopologyStrategy.

<replication\_option> e <option\_value>: Esses pares chave-valor dependem da estratégia de replicação escolhida.

Exemplos

1. Usando SimpleStrategy

O exemplo a seguir cria um keyspace chamado TestDB usando a estratégia de replicação SimpleStrategy com um fator de replicação de 1:

CREATE KEYSPACE TestDB

WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication\_factor' : 1 };

Esta configuração é geralmente usada apenas para testes ou ambientes de desenvolvimento.

2. Usando NetworkTopologyStrategy

O exemplo a seguir cria um keyspace chamado ProdDB usando a estratégia de replicação NetworkTopologyStrategy. Suponhamos que você tenha dois datacenters (DC1 e DC2), e você deseja ter um fator de replicação de 3 em DC1 e 2 em DC2:

CREATE KEYSPACE ProdDB

WITH REPLICATION = { 'class' : 'NetworkTopologyStrategy', 'DC1' : 3, 'DC2' : 2 };

Esta configuração é mais adequada para produção e cenários onde a alta disponibilidade é necessária.

Depois de criar o keyspace, você pode usá-lo com o comando:

USE TestDB;

## Teste seu conhecimento

1. Crie um Keyspace chamado Exercicio para armazenar tabelas para este estudo. Utilize o SimpleStrategy como estratégia de replicação com um fator de replicação de 1. Em seguida, use a Keyspace Exercicio.

# Tipos de dados

No Cassandra, você tem diversas opções de tipos de dados que podem ser usadas ao criar tabelas. Abaixo estão alguns dos tipos de dados mais comuns e suas explicações:

Tipos Numéricos

|  |  |
| --- | --- |
| INT | Um inteiro de 32 bits. |
| BIGINT | Um inteiro de 64 bits. |
| SMALLINT | Um inteiro de 16 bits. |
| TINYINT | Um inteiro de 8 bits. |
| VARINT | Um inteiro de precisão arbitrária. |
| DECIMAL | Um número decimal de precisão arbitrária. |
| FLOAT | Um ponto flutuante de precisão simples (32 bits). |
| DOUBLE | Um ponto flutuante de precisão dupla (64 bits). |

Tipos de Texto

|  |  |
| --- | --- |
| ASCII | Strings ASCII. |
| TEXT / VARCHAR | Strings UTF-8. |

Tipos de Data e Tempo

|  |  |
| --- | --- |
| TIMESTAMP | Uma marca de tempo (timestamp). |
| DATE | Uma data, sem hora. |
| TIME | Um tempo do dia, representado em nanossegundos. |

Tipos UUID

|  |  |
| --- | --- |
| UUID | Um identificador único universal (UUID) de 128 bits. |
| TIMEUUID | Um UUID baseado em tempo. |

Tipos Booleanos

|  |  |
| --- | --- |
| BOOLEAN | Um valor verdadeiro ou falso |

Tipos de Blob

|  |  |
| --- | --- |
| BLOB | Uma sequência de bytes (por exemplo, para armazenar arquivos binários). |

Tipos de Lista, Conjunto e Mapa

|  |  |
| --- | --- |
| LIST<T> | Uma lista ordenada de elementos do tipo T. |
| SET<T> | Um conjunto de elementos únicos do tipo T. |
| MAP<K, V> | Um mapa de chaves para valores, onde K é o tipo da chave e V é o tipo do valor. |

Tipos Compostos

|  |  |
| --- | --- |
| TUPLE<T1, T2, ...> | Um tipo de dado que agrupa múltiplos campos. |
| UDT (User-Defined Types) | Um tipo definido pelo usuário que representa uma estrutura com múltiplos campos. |

Exemplo de Criação de Tabela Usando Vários Tipos de Dados

CREATE TABLE IF NOT EXISTS exemplo (

id UUID PRIMARY KEY,

nome TEXT,

idade INT,

salario DECIMAL,

ativo BOOLEAN,

anexos BLOB,

tags SET<TEXT>,

notas MAP<TIMESTAMP, TEXT>

);

Neste exemplo, a tabela exemplo usa vários tipos de dados. id é um UUID, nome é um texto, idade é um inteiro, salario é um decimal, ativo é um booleano, anexos é um blob, tags é um conjunto de textos, e notas é um mapa de marcas de tempo para textos.

Para popular a tabela exemplo com alguns registros, você pode usar o comando INSERT. Abaixo estão alguns exemplos de como você pode inserir dados na tabela:

-- Inserindo o primeiro registro

INSERT INTO exemplo (id, nome, idade, salario, ativo, anexos, tags, notas)

VALUES (uuid(), 'Alice', 30, 50000.75, true, textAsBlob('anexo1'), {'tag1', 'tag2'}, { '2023-01-01 12:00:00' : 'Nota 1' });

-- Inserindo o segundo registro

INSERT INTO exemplo (id, nome, idade, salario, ativo, anexos, tags, notas)

VALUES (uuid(), 'Bob', 40, 60000.50, false, textAsBlob('anexo2'), {'tag3', 'tag4'}, { '2023-02-01 12:00:00' : 'Nota 2' });

-- Inserindo o terceiro registro

INSERT INTO exemplo (id, nome, idade, salario, ativo, anexos, tags, notas)

VALUES (uuid(), 'Catherine', 35, 70000.25, true, textAsBlob('anexo3'), {'tag1', 'tag5'}, { '2023-03-01 12:00:00' : 'Nota 3' });

Explicação:

* uuid(): Gera um novo UUID. Este é um recurso útil do CQL para gerar UUIDs.
* 'Alice', 'Bob', 'Catherine': São os nomes dos indivíduos.
* 30, 40, 35: São as idades.
* 50000.75, 60000.50, 70000.25: São os salários.
* true, false, true: São valores booleanos para o campo ativo.
* textAsBlob('anexo1'), textAsBlob('anexo2'), textAsBlob('anexo3'): São blobs de texto para o campo anexos. A função textAsBlob() converte um texto em blob.
* {'tag1', 'tag2'}, {'tag3', 'tag4'}, {'tag1', 'tag5'}: São conjuntos de tags.
* { '2023-01-01 12:00:00' : 'Nota 1' }: São mapas de marcas de tempo para notas.

Para verificar se os registros foram inseridos com sucesso, você pode usar o comando SELECT:

SELECT \* FROM exemplo;

# Create table

Em Cassandra, o comando CREATE TABLE é usado para criar uma tabela dentro de um keyspace específico. Cada tabela em Cassandra é uma coleção de linhas ordenadas por uma chave primária, que é composta por uma ou mais colunas. As colunas em uma tabela podem ser de vários tipos, como texto, números inteiros, UUIDs e outros.

Sintaxe Básica

A sintaxe básica para criar uma tabela em CQL é:

CREATE TABLE IF NOT EXISTS <keyspace\_name>.<table\_name> (

<column\_name> <data\_type> <column\_property>,

...,

PRIMARY KEY (<primary\_key\_column(s)>)

);

Onde:

<keyspace\_name>: O nome do keyspace onde a tabela será criada.

<table\_name>: O nome da tabela que você deseja criar.

<column\_name>: O nome da coluna na tabela.

<data\_type>: O tipo de dados da coluna (por exemplo, INT, TEXT, UUID).

<column\_property>: Propriedades adicionais da coluna (por exemplo, STATIC).

<primary\_key\_column(s)>: A coluna ou as colunas que compõem a chave primária.

Vamos criar uma tabela chamada estudantes no keyspace Escola, que contém os seguintes campos:

id (UUID): Um identificador único para cada estudante.

nome (TEXT): O nome do estudante.

idade (INT): A idade do estudante.

CREATE TABLE IF NOT EXISTS estudantes (

id UUID PRIMARY KEY,

nome TEXT,

idade INT

);

Neste exemplo, id é a chave primária, o que significa que cada linha pode ser identificada de forma única por essa coluna.

Em Cassandra, a chave primária é composta pela chave de partição e opcionalmente uma ou mais chaves de cluster. Essas componentes da chave primária servem a diferentes propósitos e têm diferentes impactos sobre o desempenho, distribuição de dados e capacidade de consulta.

**Chave de Partição:**

* Distribuição de Dados: A chave de partição determina como os dados são distribuídos em diferentes nós do cluster Cassandra. Todos os registros com a mesma chave de partição são armazenados juntos em um único nó.
* Obrigatório: Toda tabela deve ter pelo menos uma coluna como chave de partição.
* Pesquisa: As consultas em Cassandra geralmente devem especificar a chave de partição para serem eficientes.
* Sem Ordenação: Os dados em uma partição não são ordenados pela chave de partição.
* Imutabilidade: A chave de partição de um registro existente não pode ser alterada.

**Chave de Cluster**:

* Ordenação de Dados: Dentro de uma única partição, os registros são ordenados com base no valor da chave de cluster.
* Opcional: Chaves de cluster são opcionais. Você pode ter uma tabela sem nenhuma chave de cluster, embora isso limite a gama de consultas que você pode executar eficientemente.
* Pesquisa: Você pode fazer consultas eficientes usando a chave de cluster, mas geralmente apenas depois de ter restringido a chave de partição.
* Ordenação: Os dados são fisicamente armazenados na ordem da chave de cluster, o que permite consultas eficientes com operadores de comparação como <, <=, >, >=.
* Imutabilidade: As chaves de cluster de um registro existente não podem ser alteradas.

Exemplo:

CREATE TABLE exemplo (

chave\_particao TEXT,

chave\_cluster INT,

algum\_dado TEXT,

PRIMARY KEY (chave\_particao, chave\_cluster)

);

Neste exemplo, chave\_particao é a chave de partição e chave\_cluster é a chave de cluster. Todos os registros com a mesma chave\_particao seriam armazenados juntos em ordem ascendente de chave\_cluster.

## Teste seu conhecimento

1. Na Keyspace Exercicio, crie uma tabela chamada usuarios que terá os seguintes campos:

id do tipo numérico que também será a chave primária

uid do tipo UUID

nome do tipo TEXT

email do tipo TEXT

Insira dados na tabela usuários. Insira pelo menos dois registros com informações distintas. Utilize a função uuid() para gerar um UUID para o campo uid.

Execute uma consulta para exibir todos os registros da tabela usuarios. Analise os resultados.

# UPDATE

O comando UPDATE em CQL (Cassandra Query Language) é usado para modificar os valores existentes em uma ou mais colunas de uma linha específica em uma tabela. Para executar um UPDATE, você deve fornecer a chave primária da linha que deseja modificar, bem como os novos valores para as colunas que deseja atualizar.

Exemplo:

Vamos supor que você tenha uma tabela chamada estudantes, que contém as seguintes colunas:

id (UUID): Identificador único para cada estudante.

nome (TEXT): Nome do estudante.

idade (INT): Idade do estudante.

Primeiro, crie a tabela e insira alguns dados iniciais:

-- Criando a tabela estudantes

CREATE TABLE IF NOT EXISTS estudantes (

id UUID PRIMARY KEY,

nome TEXT,

idade INT

);

-- Inserindo dados

INSERT INTO estudantes (id, nome, idade) VALUES (uuid(), 'Maria', 20);

INSERT INTO estudantes (id, nome, idade) VALUES (uuid(), 'João', 22);

Agora, vamos supor que você deseja atualizar a idade de Maria para 21. Primeiro, você precisa encontrar o id do registro de Maria usando uma consulta SELECT:

SELECT \* FROM estudantes WHERE nome='Maria';

Vamos supor que o UUID retornado seja 12345678-aaaa-bbbb-cccc-1234567890ab. Agora você pode usar esse id para atualizar a idade de Maria:

UPDATE estudantes SET idade = 21 WHERE id = 12345678-aaaa-bbbb-cccc-1234567890ab;

Isso atualizará a coluna idade do registro cujo id é 12345678-aaaa-bbbb-cccc-1234567890ab, definindo a nova idade como 21.

A chave primária (id neste caso) deve ser fornecida na cláusula WHERE para identificar de forma única qual linha deve ser atualizada.

Você também pode atualizar múltiplas colunas ao mesmo tempo, como mostrado abaixo:

UPDATE estudantes SET idade = 21, nome = 'Maria Silva' WHERE id = 12345678-aaaa-bbbb-cccc-1234567890ab;

## Teste seu conhecimento

1. Escolha um registro e atualize o campo nome. Verifique se a atualização foi bem-sucedida executando uma consulta.
2. Escolha um registro e atualize o nome e o e-mail. Verifique se a atualização foi bem-sucedida executando uma consulta.

# DELETE

O comando DELETE em Cassandra CQL é utilizado para remover um ou mais registros de uma tabela. O comando pode ser usado de várias formas: para deletar colunas específicas de um registro, para deletar todo um registro, ou até para deletar vários registros que satisfaçam uma determinada condição. É importante observar que, ao contrário de bancos de dados relacionais, você não pode executar um DELETE sem condições para remover todos os registros de uma tabela; você precisa especificar pelo menos uma condição de chave primária.

Sintaxe Básica

-- Deleta todo o registro com uma chave primária específica

DELETE FROM <table\_name> WHERE primary\_key\_condition;

-- Deleta colunas específicas de um registro com uma chave primária específica

DELETE column1, column2 FROM <table\_name> WHERE primary\_key\_condition;

Onde:

<table\_name>: O nome da tabela de onde você deseja deletar o registro.

primary\_key\_condition: A condição que define qual registro deve ser deletado. Isso geralmente envolve fornecer um valor para cada parte da chave primária da tabela.

Exemplo

Suponha que você tenha uma tabela chamada estudantes com a seguinte estrutura:

CREATE TABLE IF NOT EXISTS estudantes (

id UUID PRIMARY KEY,

nome TEXT,

idade INT

);

Você pode inserir alguns dados usando o comando INSERT:

INSERT INTO estudantes (id, nome, idade) VALUES (uuid(), 'Alice', 20);

INSERT INTO estudantes (id, nome, idade) VALUES (uuid(), 'Bob', 22);

Se você quiser deletar o registro do estudante com um id específico, você pode usar o seguinte comando:

DELETE FROM estudantes WHERE id = 12345678-1234-5678-1234-567812345678;

Se você quiser deletar apenas a coluna idade do estudante com um id específico, você pode fazer isso da seguinte forma:

DELETE idade FROM estudantes WHERE id = 12345678-1234-5678-1234-567812345678;

Isso removerá apenas o valor da coluna idade para o estudante com o id especificado, mantendo o resto do registro intacto.

## Teste seu conhecimento

1. Insira dados na tabela usuários. Insira pelo menos dois registros com informações distintas. Utilize a função uuid() para gerar um UUID para o campo uid.
2. Escolha um registro para excluir da tabela usuarios. Execute uma consulta para garantir que o registro foi removido.
3. Escolha uma registro da tabela usuários, em seguida, exclua uma coluna desse registro. Execute uma consulta para garantir que a coluna foi removida.

# Operadores de Comparação

O Cassandra CQL suporta um conjunto limitado de operadores de comparação na cláusula WHERE para filtrar registros durante consultas SELECT. Diferente de bancos de dados relacionais tradicionais como o SQL Server ou MySQL, o Cassandra tem restrições mais rígidas devido ao seu design focado em escalabilidade horizontal.

**Devido às restrições de design do Cassandra, você só pode usar operadores de comparação com colunas que são parte da chave primária ou têm um índice secundário.**

Operadores de Comparação Suportados

Igual (=): Compara se o valor é igual ao especificado.

SELECT \* FROM table\_name WHERE column\_name = value;

Maior Que (>) e Menor Que (<): Compara se o valor é maior/menor que o especificado.

SELECT \* FROM table\_name WHERE column\_name > value;

SELECT \* FROM table\_name WHERE column\_name < value;

Maior ou Igual (>=) e Menor ou Igual (<=): Compara se o valor é maior/menor ou igual ao especificado.

SELECT \* FROM table\_name WHERE column\_name >= value;

SELECT \* FROM table\_name WHERE column\_name <= value;

**Diferente (!= ou <>): Embora muitos bancos de dados relacionais suportem operadores de "diferente", o Cassandra não suporta esse operador de comparação.**

IN: Utilizado para comparar se o valor está dentro de um conjunto de valores especificados.

SELECT \* FROM table\_name WHERE column\_name IN (value1, value2, ...);

# Operadores de Lógicos

Operador AND

Em Cassandra CQL, o operador AND é usado para combinar múltiplas condições na cláusula WHERE e todos os critérios devem ser verdadeiros para que o registro seja selecionado ou modificado. No entanto, o uso de AND é mais restrito em Cassandra em comparação com bancos de dados relacionais tradicionais. As restrições são devidas à forma como o Cassandra armazena e recupera dados para otimizar a leitura e a gravação em escala.

Sintaxe Básica com AND:

SELECT \* FROM table\_name WHERE condition1 AND condition2;

Operador OR

O operador OR **não é suportado** na cláusula WHERE em Cassandra CQL. Isso significa que você não pode usar OR para combinar múltiplas condições na mesma cláusula WHERE. Isso é uma limitação do Cassandra que é importante para garantir que as operações de leitura sejam eficientes e previsíveis em termos de desempenho.

Exemplo Usando AND

Suponhamos que temos uma tabela chamada pedidos com a seguinte estrutura:

CREATE TABLE IF NOT EXISTS pedidos (

id UUID,

cliente\_id UUID,

valor DECIMAL,

data TIMESTAMP,

PRIMARY KEY (cliente\_id, data)

);

Aqui, a chave primária é composta de cliente\_id como a chave de partição e data como a chave de cluster.

SELECT \* FROM pedidos WHERE cliente\_id = 1234-5678-9101-1121 AND

data >= '2022-01-01 00:00:00' AND

data <= '2022-12-31 23:59:59';

Neste exemplo, estamos usando AND para combinar três condições:

cliente\_id deve ser igual a 1234-5678-9101-1121.

data deve ser maior ou igual a '2022-01-01 00:00:00'.

data deve ser menor ou igual a '2022-12-31 23:59:59'.

Note que todas essas condições são aplicadas a colunas que fazem parte da chave primária composta (cliente\_id e data).

## Teste seu conhecimento

Vamos criar uma tabela chamada produtos. Esta tabela terá um campo categoria como chave de partição e um campo preco como chave de cluster.

Criação da Tabela

CREATE TABLE IF NOT EXISTS produtos (

categoria TEXT,

nome TEXT,

preco DECIMAL,

disponivel BOOLEAN,

PRIMARY KEY (categoria, preco)

);

Vamos inserir alguns dados de exemplo na tabela:

INSERT INTO produtos (categoria, nome, preco, disponivel) VALUES ('Eletrônicos', 'Celular', 999.99, TRUE);

INSERT INTO produtos (categoria, nome, preco, disponivel) VALUES ('Eletrônicos', 'Notebook', 1499.99, TRUE);

INSERT INTO produtos (categoria, nome, preco, disponivel) VALUES ('Roupas', 'Camiseta', 19.99, TRUE);

INSERT INTO produtos (categoria, nome, preco, disponivel) VALUES ('Roupas', 'Calça', 49.99, FALSE);

INSERT INTO produtos (categoria, nome, preco, disponivel) VALUES ('Alimentos', 'Pão', 1.99, TRUE);

INSERT INTO produtos (categoria, nome, preco, disponivel) VALUES ('Alimentos', 'Leite', 2.99, TRUE);

1. Selecione todos os produtos na categoria 'Eletrônicos'.
2. Selecione todos os produtos na categoria 'Roupas' com preço menor que 50.
3. Selecione todos os produtos na categoria 'Alimentos' com preço maior ou igual a 2.
4. Selecione todos os produtos na categoria 'Eletrônicos' que estão disponíveis. (Aqui é esperado um erro por conta do operado AND)
5. Selecione todos os produtos na categoria 'Roupas' com preço entre 20 e 50.

# Operador LIKE

Em Cassandra, o uso da indexação customizada com SASI (SSTable-Attached Secondary Index) pode permitir que você realize consultas usando o operador LIKE de uma forma mais eficiente do que com índices secundários padrão.

A sintaxe básica para criar um índice SASI é:

CREATE CUSTOM INDEX IF NOT EXISTS index\_name ON table\_name (column\_name)

USING 'org.apache.cassandra.index.sasi.SASIIndex'

WITH OPTIONS = {'option\_name': 'option\_value', ...};

Exemplo

Suponha que temos uma tabela usuarios com uma coluna nome e gostaríamos de fazer buscas usando o operador LIKE.

Criação da Tabela:

CREATE TABLE IF NOT EXISTS usuarios (

id UUID PRIMARY KEY,

nome TEXT,

idade INT

);

Criando o Índice SASI:

CREATE CUSTOM INDEX IF NOT EXISTS nome\_sasi\_idx ON usuarios (nome)

USING 'org.apache.cassandra.index.sasi.SASIIndex'

WITH OPTIONS = {'analyzed': 'true', 'analyzer\_class': 'org.apache.cassandra.index.sasi.analyzer.StandardAnalyzer', 'tokenization\_enable\_stemming': 'true'};

Neste exemplo, o índice nome\_sasi\_idx é criado na coluna nome da tabela usuarios. Os parâmetros analyzed e analyzer\_class são opções que você pode fornecer para customizar o comportamento do índice SASI.

Inserir alguns dados

INSERT INTO usuarios (id, nome, idade) VALUES (uuid(), 'Alice', 30);

INSERT INTO usuarios (id, nome, idade) VALUES (uuid(), 'Bob', 40);

INSERT INTO usuarios (id, nome, idade) VALUES (uuid(), 'Charlie', 50);

Realizando uma consulta com LIKE:

SELECT \* FROM usuarios WHERE nome LIKE 'Al%';

Esta consulta retornará todos os registros onde o campo nome começa com 'Al', graças ao índice SASI customizado que criamos.

Lembre-se de que a criação de índices customizados como o SASI pode afetar o desempenho e a manutenção do seu cluster Cassandra, portanto, teste cuidadosamente antes de aplicar em ambientes de produção.

## Teste seu conhecimento

Criação da Tabela:

CREATE TABLE IF NOT EXISTS pessoas (

id UUID PRIMARY KEY,

nome TEXT,

sobrenome TEXT,

email TEXT,

profissao TEXT

);

Vamos criar um índice SASI na coluna nome para facilitar a busca de pessoas pelo nome usando o operador LIKE.

CREATE CUSTOM INDEX IF NOT EXISTS nome\_sasi\_idx ON pessoas (nome)

USING 'org.apache.cassandra.index.sasi.SASIIndex'

WITH OPTIONS = {'analyzed': 'true', 'analyzer\_class': 'org.apache.cassandra.index.sasi.analyzer.StandardAnalyzer', 'tokenization\_enable\_stemming': 'true'};

Inserindo alguns dados:

INSERT INTO pessoas (id, nome, sobrenome, email, profissao) VALUES (uuid(), 'Alice', 'Johnson', 'alice@email.com', 'Engenheira');

INSERT INTO pessoas (id, nome, sobrenome, email, profissao) VALUES (uuid(), 'Bob', 'Smith', 'bob@email.com', 'Programador');

INSERT INTO pessoas (id, nome, sobrenome, email, profissao) VALUES (uuid(), 'Carol', 'Williams', 'carol@email.com', 'Designer');

INSERT INTO pessoas (id, nome, sobrenome, email, profissao) VALUES (uuid(), 'Dave', 'Brown', 'dave@email.com', 'Engenheiro');

INSERT INTO pessoas (id, nome, sobrenome, email, profissao) VALUES (uuid(), 'Eva', 'Davis', 'eva@email.com', 'Médica');

Use o operador LIKE para executar os exercícios abaixo. Indique quais não são possíveis de serem realizados mesmo com o índice SASI.

1. Encontrar todas as pessoas cujo nome comece com "Al"
2. Encontrar todas as pessoas cujo nome termine com "ce"
3. Encontrar todas as pessoas cujo nome contém a substring "ro"
4. Encontrar todas as pessoas cuja profissão termina com "or"
5. Encontrar todas as pessoas cujo e-mail comece com "d"

# UDF

Em Cassandra, UDFs (User-Defined Functions) são funções personalizadas que você pode criar para realizar operações específicas que não são suportadas pelas funções internas do Cassandra.

Função para dobrar um valor inteiro

Vamos criar uma função simples chamada double\_int que recebe um valor inteiro e retorna o seu dobro.

CREATE OR REPLACE FUNCTION double\_int (input int)

RETURNS NULL ON NULL INPUT

RETURNS int

LANGUAGE java

AS $$ return input \* 2; $$;

Você pode testar a função da seguinte forma:

SELECT double\_int(1024) FROM estudantes;

Função para calcular o comprimento de uma string

Vamos criar uma função chamada string\_length que retorna o comprimento de uma string.

CREATE OR REPLACE FUNCTION string\_length(s text)

RETURNS NULL ON NULL INPUT

RETURNS int

LANGUAGE java

AS $$ return s.length(); $$;

Você pode testar a função da seguinte forma:

SELECT string\_length(nome) FROM estudantes;

Opções para valores NULL:

RETURNS NULL ON NULL INPUT: Se qualquer argumento da função for NULL, a função retornará NULL.

CALLED ON NULL INPUT: Mesmo que qualquer argumento da função seja NULL, a função ainda será chamada.

Idioma:

O idioma mais comum para escrever UDFs é o Java, visto que o Cassandra é escrito em Java. Você deve fornecer o código-fonte da função como uma string (entre $$ no exemplo).

## Teste seu conhecimento

1. Crie uma UDF denominada de ‘quadrado’ que aceita um número do tipo double e retorna a raiz quadrada do mesmo, a entrada de dados deve ser denominada de “input”. O código JAVA para essa operação é

$$ return Math.sqrt(input); $$;

1. Crie uma UDF chamada to\_upper que aceita uma string e retorna essa string convertida para maiúsculas, a entrada de dados deve ser denominada de input. O código JAVA para essa operação é

$$ return input.toUpperCase(); $$;

1. Teste as duas UDFS criadas.

# Exercícios

1. Crie um keyspace chamado companyDB com um fator de replicação de 1 usando o SimpleStrategy.
2. Dentro do keyspace companyDB, crie uma tabela chamada employees com os seguintes campos: emp\_id (UUID), first\_name (TEXT), last\_name (TEXT), age (INT), e salary (DECIMAL). Escolha quais campos serão chave primária e quais serão chave de cluster.
3. Insira 3 registros na tabela employees.
4. Selecione todos os registros da tabela employees onde a idade é maior que 30.
5. Aumente o salário dos funcionários com age maior que 40 em 10%.
6. Delete registros da tabela employees onde o salary é menor que 5000.
7. Crie um índice na coluna last\_name da tabela employees.
8. Selecione registros da tabela employees onde o last\_name começa com "Smi".
9. Calcule o salário médio dos funcionários da tabela employees.
10. Calcule a soma de todos os salários dos funcionários da tabela employees