estágio  
Desenvolvimento Full Stack

Aluna de Engenharia de Software

linha horizontal

# 16 e 17 de setembro de 2024

## Enunciado do Exercício

Criar um backend com um cron job que realiza as seguintes operações:

1. **Buscar Dados**: Buscar todos os registros de uma tabela (tabela1) em um banco de dados (db1).
2. **Copiar Dados**: Copiar todos esses registros para outra tabela (tabela2) em um banco de dados diferente (db2).

**Tecnologias**: Utilizar JavaScript com Express e MySQL.

**Nome do Projeto**: cron-job-backend

Um cron job é uma tarefa agendada que é executada automaticamente em intervalos de tempo específicos. No seu caso, o cron job será responsável por buscar dados de uma tabela e copiá-los para outra tabela, de maneira automática.

## Fase 1: Configuração do projeto

**Criação da pasta do projeto e inicialização do Node.js**: Aqui, você cria um novo projeto Node.js, o que serve como uma base para o desenvolvimento de aplicações em JavaScript.

* mkdir cron-job-backend: Cria uma nova pasta para o seu projeto.
* npm init -y: Inicializa o projeto e cria um arquivo package.json, que guarda informações do seu projeto e suas dependências.

### Instalação das Dependências

Instale as bibliotecas necessárias:

npm install express mysql2 node-cron

* **express**: Framework para criar servidores web com facilidade.
* **mysql2**: Biblioteca para interagir com bancos de dados MySQL.
* **node-cron**: Biblioteca para criar cron jobs em Node.js.

### Passo 2: Crie a estrutura de arquivos

Na raiz do projeto, crie:

* index.js: para configurar o servidor e cron job.

### Passo 3. Código Atualizado para index.js

const express = require('express');

const mysql = require('mysql2');

const cron = require('node-cron');

const app = express();

// Configurar os bancos de dados MySQL

const db1 = mysql.createConnection({

host: 'localhost',

user: 'root',

password: 'your\_password',

database: 'db1'

});

const db2 = mysql.createConnection({

host: 'localhost',

user: 'root',

password: 'your\_password',

database: 'db2'

});

// Conectar aos bancos de dados

db1.connect(err => {

if (err) {

console.error('Erro ao conectar ao banco de dados db1:', err);

return;

}

console.log('Conectado ao banco de dados db1.');

});

db2.connect(err => {

if (err) {

console.error('Erro ao conectar ao banco de dados db2:', err);

return;

}

console.log('Conectado ao banco de dados db2.');

});

// Criar tabelas exemplo no banco de dados 1 e 2

db1.query('CREATE TABLE IF NOT EXISTS tabela1 (id INT PRIMARY KEY, nome VARCHAR(255))');

db2.query('CREATE TABLE IF NOT EXISTS tabela2 (id INT PRIMARY KEY, nome VARCHAR(255))');

// Função para copiar dados de tabela1 (db1) para tabela2 (db2)

const copiarDados = () => {

// Obter todas as colunas de tabela1

db1.query('DESCRIBE tabela1', (err, columns) => {

if (err) {

console.error('Erro ao obter informações da tabela:', err);

return;

}

const colunas = columns.map(col => col.Field).join(', '); // Ex: "id, nome"

const placeholders = columns.map(() => '?').join(', '); // Ex: "?, ?"

const updatePlaceholders = columns.map(col => `${col.Field} = VALUES(${col.Field})`).join(', '); // Ex: "id = VALUES(id), nome = VALUES(nome)"

// Buscar dados da tabela1

db1.query('SELECT \* FROM tabela1', (err, rows) => {

if (err) {

console.error('Erro ao buscar dados:', err);

return;

}

rows.forEach((row) => {

const values = columns.map(col => row[col.Field]); // Obter valores dinamicamente

// Inserir ou atualizar dados na tabela2

db2.query(

`INSERT INTO tabela2 (${colunas}) VALUES (${placeholders}) ON DUPLICATE KEY UPDATE ${updatePlaceholders}`,

values,

(err) => {

if (err) {

console.error('Erro ao inserir ou atualizar dados:', err);

} else {

console.log(`Dados copiados: ${row.id}`);

}

}

);

});

});

});

};

// Agendar cron job para rodar a cada minuto

cron.schedule('\* \* \* \* \*', () => {

console.log('Executando cron job...');

copiarDados();

});

// Servidor básico

app.get('/', (req, res) => {

res.send('Cron job está rodando!');

});

app.listen(3000, () => {

console.log('Servidor rodando na porta 3000');

});

### Explicação Detalhada

#### **1. Importação das Bibliotecas**

const express = require('express');

const mysql = require('mysql2');

const cron = require('node-cron');

**express**: Biblioteca para criar o servidor web.

**mysql2**: Biblioteca para interagir com MySQL.

**node-cron**: Biblioteca para agendar tarefas.

#### **2. Inicialização do Servidor Express**

const app = express();

**app**: Cria uma instância do aplicativo Express, que será usada para definir rotas e iniciar o servidor.

#### **3. Configuração dos Bancos de Dados**

const db1 = mysql.createConnection({

host: 'localhost',

user: 'root',

password: 'your\_password',

database: 'db1'

});

const db2 = mysql.createConnection({

host: 'localhost',

user: 'root',

password: 'your\_password',

database: 'db2'

});

Cria conexões para os bancos de dados db1 e db2.

#### **4. Criação das Tabelas**

db1.query('CREATE TABLE IF NOT EXISTS tabela1 (id INT PRIMARY KEY, nome VARCHAR(255))');

db2.query('CREATE TABLE IF NOT EXISTS tabela2 (id INT PRIMARY KEY, nome VARCHAR(255))');

**db1.query e db2.query**: Executa comandos SQL para criar tabelas se elas não existirem. tabela1 e tabela2 têm colunas id e nome.

**CÓDIGO INTEIRO**

const express = require('express');

const mysql = require('mysql');

const cron = require('node-cron');

const app = express();

// Configuração do banco de dados MySQL

const db1 = mysql.createConnection({

host: 'localhost',

user: 'root',

password: 'password',

database: 'db1'

});

const db2 = mysql.createConnection({

host: 'localhost',

user: 'root',

password: 'password',

database: 'db2'

});

// Conectar aos bancos de dados

db1.connect((err) => {

if (err) {

console.error('Erro ao conectar ao banco de dados db1:', err);

return;

}

console.log('Conectado ao banco de dados db1');

});

db2.connect((err) => {

if (err) {

console.error('Erro ao conectar ao banco de dados db2:', err);

return;

}

console.log('Conectado ao banco de dados db2');

});

// Criação das tabelas de exemplo

db1.query('CREATE TABLE IF NOT EXISTS tabela1 (id INT PRIMARY KEY, nome VARCHAR(255))', (err) => {

if (err) {

console.error('Erro ao criar tabela1:', err);

}

});

db2.query('CREATE TABLE IF NOT EXISTS tabela2 (id INT PRIMARY KEY, nome VARCHAR(255))', (err) => {

if (err) {

console.error('Erro ao criar tabela2:', err);

}

});

// Função para copiar dados de tabela1 (db1) para tabela2 (db2)

const copiarDados = () => {

// Obter informações das colunas de tabela1

db1.query('SHOW COLUMNS FROM tabela1', (err, columns) => {

if (err) {

console.error('Erro ao obter informações das colunas de tabela1:', err);

return;

}

// Preparar as partes da consulta SQL dinamicamente

const colunas = columns.map(col => col.Field).join(', ');

const placeholders = columns.map(() => '?').join(', ');

const updatePlaceholders = columns.map(col => `${col.Field} = VALUES(${col.Field})`).join(', ');

// Consultar todos os dados de tabela1

db1.query('SELECT \* FROM tabela1', (err, rows) => {

if (err) {

console.error('Erro ao buscar dados de tabela1:', err);

return;

}

rows.forEach((row) => {

// Preparar os valores para inserção

const values = columns.map(col => row[col.Field]);

// Inserir ou atualizar os dados em tabela2

db2.query(

`INSERT INTO tabela2 (${colunas}) VALUES (${placeholders}) ON DUPLICATE KEY UPDATE ${updatePlaceholders}`,

values,

(err) => {

if (err) {

console.error('Erro ao inserir ou atualizar dados em tabela2:', err);

} else {

console.log(`Dados copiados: ${row.id}`);

}

}

);

});

});

});

};

// Agendar o cron job para rodar a cada minuto

cron.schedule('\* \* \* \* \*', () => {

console.log('Executando cron job...');

copiarDados();

});

// Servidor básico

app.get('/', (req, res) => {

res.send('Cron job está rodando!');

});

app.listen(3000, () => {

console.log('Servidor rodando na porta 3000');

});

### Explicação Detalhada

### **Função copiarDados**

* + **Obtenção das colunas:**

db1.query('SHOW COLUMNS FROM tabela1', (err, columns) => {

if (err) {

console.error('Erro ao obter informações das colunas de tabela1:', err);

return;

}

});

* + SHOW COLUMNS FROM tabela1: Consulta SQL que retorna informações sobre as colunas da tabela tabela1. Isso inclui o nome da coluna (Field) e outros detalhes.
  + O callback recebe columns, que é uma lista de objetos, cada um representando uma
  + **Preparação das Strings SQL**:

const colunas = columns.map(col => col.Field).join(', ');

const placeholders = columns.map(() => '?').join(', ');

const updatePlaceholders = columns.map(col => `${col.Field} = VALUES(${col.Field})`).join(', ');

* + - colunas: Cria uma string com os nomes das colunas separados por vírgulas, que será usada na cláusula INSERT INTO.
    - placeholders: Cria uma string com placeholders (?) para os valores que serão inseridos. Cada ? será substituído por um valor real na consulta.
    - updatePlaceholders: Cria uma string para a cláusula ON DUPLICATE KEY UPDATE. Esta parte define como atualizar os valores das colunas em caso de conflito com uma chave primária existente.
  + **Consulta e Inserção/Atualização**:

db1.query('SELECT \* FROM tabela1', (err, rows) => {

if (err) {

console.error('Erro ao buscar dados de tabela1:', err);

return;

}

rows.forEach((row) => {

const values = columns.map(col => row[col.Field]);

db2.query(

`INSERT INTO tabela2 (${colunas}) VALUES (${placeholders}) ON DUPLICATE KEY UPDATE ${updatePlaceholders}`,

values,

(err) => {

if (err) {

console.error('Erro ao inserir ou atualizar dados em tabela2:', err);

} else {

console.log(`Dados copiados: ${row.id}`);

}

}

);

});

});

* + - SELECT \* FROM tabela1: Consulta SQL que retorna todos os registros da tabela tabela1.
    - rows.forEach((row) => { ... }): Itera sobre cada registro retornado da tabela tabela1.
    - const values = columns.map(col => row[col.Field]);: Prepara um array de valores para o registro atual. Os valores são extraídos com base nos nomes das colunas.
    - db2.query(...): Executa a consulta SQL para inserir ou atualizar os dados na tabela tabela2. O comando SQL é construído dinamicamente para incluir os nomes das colunas, placeholders e a cláusula ON DUPLICATE KEY UPDATE.

### Funcionamento do ON DUPLICATE KEY UPDATE

A cláusula ON DUPLICATE KEY UPDATE é uma característica do MySQL que permite que a mesma instrução SQL faça tanto a inserção quanto a atualização. Se um registro com a mesma chave primária (id, neste caso) já existir em tabela2, o MySQL atualizará o registro existente com os novos valores. Caso contrário, um novo registro será inserido.

**A função copiarDados:**

* Obtém informações das colunas de tabela1.
* Prepara as partes da consulta SQL para inserir ou atualizar dados de maneira dinâmica.
* Consulta todos os registros de tabela1.
* Insere ou atualiza cada registro em tabela2 com base na existência de uma chave primária.

Essas etapas garantem que os dados sejam sincronizados corretamente entre tabela1 e tabela2 sem a necessidade de escrever SQL específico para cada tabela.

1. **Agendamento do Cron Job**
   * O cron job é configurado para executar a função copiarDados a cada minuto.
2. **Servidor Express**
   * Configura um servidor básico para confirmar que o cron job está rodando.

***Essa versão do código é adequada para MySQL e garante que os dados sejam inseridos ou atualizados dinamicamente na tabela de destino.***

😀😀😀😀😀😀😀😀😀😀