Aula 5

Created	@January 13, 2024 8:52 PM
Data	@January 17, 2024
Status	Iniciado
Tipo	Aula

Introdução a Banco de Dados

Um panorama sobre os tipos de banco de dados existentes (SQL e NoSQL)

Os bancos de dados podem ser categorizados em dois principais tipos: SQL (Structured Query Language) e NoSQL (Not Only SQL). Essas categorias referem-se à forma como os dados são armazenados, modelados e consultados. Vamos dar uma visão geral de ambos os tipos:

Bancos de Dados SQL:

1. **Modelo Relacional:**
    - **Características Principais:**
        - Dados organizados em tabelas relacionadas.
        - Uso de esquemas pré-definidos.
        - Uso extensivo de SQL para consultas.
        - Transações ACID (Atomicidade, Consistência, Isolamento, Durabilidade) garantem integridade dos dados.
    - **Exemplos de Bancos de Dados SQL:**

    - MySQL
    - PostgreSQL
    - Oracle
    - Microsoft SQL Server
    - SQLite
- **Aplicações Típicas:**
    - Aplicações empresariais.
    - Sistemas de gerenciamento de conteúdo.
    - Sistemas financeiros.

Bancos de Dados NoSQL:

1. **Modelo Não-Relacional:**
    - **Características Principais:**
        - Dados podem ser armazenados em diversos formatos: documentos, chave-valor, colunas, grafos, etc.
        - Esquemas dinâmicos ou ausência de esquema.
        - Consultas podem variar de acordo com o tipo de banco de dados.
    - **Tipos Comuns de Bancos de Dados NoSQL:**
        - **Documentos:** MongoDB, CouchDB.
        - **Chave-Valor:** Redis, DynamoDB.
        - **Colunas:** Cassandra, HBase.
        - **Grafos:** Neo4j, ArangoDB.
    - **Aplicações Típicas:**
        - Aplicações web modernas.
        - Sistemas de Big Data e análise.
        - Aplicações que exigem escalabilidade horizontal.

Diferenças Principais:

1. **Esquema:**
   - **SQL:** Usa um esquema rígido e pré-definido.
   - **NoSQL:** Pode ter um esquema dinâmico ou ausência de esquema.
2. **Consulta:**
   - **SQL:** Utiliza a linguagem SQL padrão para consultas.
   - **NoSQL:** As consultas podem variar dependendo do tipo de banco de dados NoSQL.
3. **Transações:**
   - **SQL:** Garante transações ACID para manter a consistência dos dados.
   - **NoSQL:** Oferece maior flexibilidade, mas as transações podem variar entre os diferentes tipos de NoSQL.
4. **Escalabilidade:**
   - **SQL:** Geralmente escalado verticalmente (mais recursos em um servidor único).
   - **NoSQL:** Projetado para escalabilidade horizontal (adicionando mais servidores).
5. **Flexibilidade de Esquema:**
   - **SQL:** Rigidez no esquema, qualquer alteração geralmente requer uma migração cuidadosa.
   - **NoSQL:** Maior flexibilidade, permite a adição de novos campos sem alterações no esquema.

A escolha entre SQL e NoSQL geralmente depende dos requisitos específicos do projeto, como a natureza dos dados, escalabilidade, flexibilidade de esquema e os tipos de consultas necessárias. Ambas as abordagens têm suas vantagens e desvantagens, e a escolha deve ser feita considerando cuidadosamente os objetivos e requisitos do sistema em questão.

Criando seus primeiros schemas com SQLite (DDL)

O SQLite é um sistema de gerenciamento de banco de dados relacional (RDBMS) incorporado, leve e de código aberto. Ele é projetado para ser incorporado em aplicativos e não exige um servidor de banco de dados separado, sendo uma biblioteca C que fornece funcionalidades completas de banco de dados relacional para aplicativos.

Aqui estão algumas características e aspectos importantes do SQLite:

1. **Incorporado e Sem Servidor:**
    - O SQLite é projetado para ser incorporado diretamente nas aplicações, o que significa que ele não requer um servidor de banco de dados separado para ser executado. O banco de dados SQLite é armazenado como um único arquivo em disco.
2. **Sem Configuração:**
    - Ao contrário de sistemas de banco de dados mais robustos que requerem configurações complexas, o SQLite não exige configurações significativas. Ele pode ser iniciado e usado com muito pouco esforço.
3. **Autossuficiente:**
    - O SQLite não depende de um processo de servidor separado. Todas as operações de leitura e gravação são realizadas diretamente no arquivo de banco de dados. Isso o torna adequado para aplicativos de desktop, aplicativos móveis e sistemas embarcados.
4. **Transações ACID:**
    - O SQLite suporta transações ACID (Atomicidade, Consistência, Isolamento, Durabilidade), garantindo a integridade dos dados mesmo em condições adversas.
5. **Tipos de Dados Dinâmicos:**
    - O SQLite permite que os tipos de dados sejam atribuídos dinamicamente, o que oferece flexibilidade na definição do esquema do banco de dados.
6. **Ampla Disponibilidade:**

- É uma biblioteca C de código aberto e amplamente adotada, estando disponível para várias plataformas e linguagens de programação.

7. **Ótimo para Pequenas Aplicações:**
   - Devido à sua simplicidade e leveza, o SQLite é frequentemente usado em aplicações menores, como aplicativos móveis, navegadores web e ferramentas de desktop.
8. **Limitações de Escalabilidade:**
   - Embora seja uma escolha excelente para aplicações menores, o SQLite pode ter limitações de escalabilidade em ambientes de alto tráfego ou grandes volumes de dados em comparação com bancos de dados mais robustos.
9. **Uso Comum:**
   - O SQLite é comumente usado em uma variedade de aplicações, desde dispositivos móveis até navegadores, ferramentas de gerenciamento de banco de dados e aplicativos de desktop.

Em resumo, o SQLite é uma escolha popular para aplicações que necessitam de um banco de dados leve e incorporado. Sua simplicidade, baixa sobrecarga e ampla adoção o tornam uma opção atraente para uma variedade de cenários de desenvolvimento.

Inserindo, atualizando e excluindo registros com SQLite (DML)

Para realizar operações de manipulação de dados em um banco de dados SQLite, você utilizará comandos SQL conhecidos como DML (Data Manipulation Language). Os principais comandos DML são `INSERT` (inserção), `UPDATE` (atualização) e `DELETE` (exclusão). Abaixo, fornecerei exemplos de como executar essas operações com o SQLite, utilizando a biblioteca `sqlite3` para Node.js. Certifique-se de ter a biblioteca instalada antes de prosseguir:

```
bashCopy code
npm install sqlite3
```

1. Inserindo Registros (`INSERT`):

```
javascriptCopy code
const sqlite3 = require('sqlite3').verbose();

// Conectar ao banco de dados (ou criar se não existir)
const db = new sqlite3.Database('exemplo.db');

// Dados a serem inseridos
const novoRegistro = {
  nome: 'John Doe',
  idade: 30,
  cidade: 'Exemploville'
};

// Comando SQL de inserção
const sqlInserir = 'INSERT INTO usuarios (nome, idade, cidad
e) VALUES (?, ?, ?)';

// Executar a inserção
db.run(sqlInserir, [novoRegistro.nome, novoRegistro.idade, no
voRegistro.cidade], function(err) {
  if (err) {
    return console.error(err.message);
  }
  console.log(`Registro inserido com ID: ${this.lastID}`);
});

// Fechar a conexão após a operação
```

```
db.close();
```

2. Atualizando Registros (`UPDATE`):

```
javascriptCopy code
const sqlite3 = require('sqlite3').verbose();

// Conectar ao banco de dados
const db = new sqlite3.Database('exemplo.db');

// Novos dados para atualização
const novosDados = {
  idade: 31,
  cidade: 'NovoVille'
};

// Comando SQL de atualização
const sqlAtualizar = 'UPDATE usuarios SET idade = ?, cidade =
? WHERE nome = ?';

// Executar a atualização
db.run(sqlAtualizar, [novosDados.idade, novosDados.cidade, 'J
ohn Doe'], function(err) {
  if (err) {
    return console.error(err.message);
  }
  console.log(`Registros atualizados: ${this.changes}`);
});

// Fechar a conexão após a operação
db.close();
```

3. Excluindo Registros (`DELETE`):

```
javascriptCopy code
const sqlite3 = require('sqlite3').verbose();

// Conectar ao banco de dados
const db = new sqlite3.Database('exemplo.db');

// Comando SQL de exclusão
const sqlExcluir = 'DELETE FROM usuarios WHERE nome = ?';

// Executar a exclusão
db.run(sqlExcluir, ['John Doe'], function(err) {
  if (err) {
    return console.error(err.message);
  }
  console.log(`Registros excluídos: ${this.changes}`);
});

// Fechar a conexão após a operação
db.close();
```

Lembre-se de adaptar os comandos SQL e os dados conforme a estrutura do seu próprio banco de dados SQLite. Além disso, é uma boa prática usar parâmetros em vez de inserir diretamente os valores no SQL para evitar vulnerabilidades de injeção de SQL. Os exemplos acima são simplificados para demonstrar os conceitos básicos.

Consultando registros com SQLite (DQL)

Para realizar consultas (DQL - Data Query Language) em um banco de dados SQLite, você utiliza o comando SQL `SELECT`. Abaixo, vou fornecer um exemplo simples de como executar consultas com o SQLite utilizando a biblioteca `sqlite3` para Node.js:

Certifique-se de ter a biblioteca instalada antes de prosseguir:

```
bashCopy code
npm install sqlite3
```

A seguir, exemplos de consultas:

1. Consultando todos os registros:

```
javascriptCopy code
const sqlite3 = require('sqlite3').verbose();

// Conectar ao banco de dados
const db = new sqlite3.Database('exemplo.db');

// Comando SQL de consulta
const sqlConsultaTodos = 'SELECT * FROM usuarios';

// Executar a consulta
db.all(sqlConsultaTodos, [], (err, rows) => {
  if (err) {
    throw err;
  }

  // Processar os resultados
  rows.forEach((row) => {
    console.log(row);
  });
});

// Fechar a conexão após a operação
db.close();
```

2. Consultando com condições:

```
javascriptCopy code
const sqlite3 = require('sqlite3').verbose();

// Conectar ao banco de dados
const db = new sqlite3.Database('exemplo.db');

// Condição de consulta
const idadeMinima = 25;

// Comando SQL de consulta com condição
const sqlConsultaComCondicao = 'SELECT * FROM usuarios WHERE
idade >= ?';

// Executar a consulta
db.all(sqlConsultaComCondicao, [idadeMinima], (err, rows) =>
{
  if (err) {
    throw err;
  }

  // Processar os resultados
  rows.forEach((row) => {
    console.log(row);
  });
});

// Fechar a conexão após a operação
db.close();
```

3. Consultando uma única linha:

```
javascriptCopy code
const sqlite3 = require('sqlite3').verbose();

// Conectar ao banco de dados
const db = new sqlite3.Database('exemplo.db');

// Nome para consulta
const nomeConsulta = 'Alice';

// Comando SQL de consulta para uma única linha
const sqlConsultaUnicaLinha = 'SELECT * FROM usuarios WHERE n
ome = ?';

// Executar a consulta
db.get(sqlConsultaUnicaLinha, [nomeConsulta], (err, row) => {
  if (err) {
    throw err;
  }

  // Processar o resultado
  console.log(row);
});

// Fechar a conexão após a operação
db.close();
```

Lembre-se de ajustar os comandos SQL e os parâmetros conforme a estrutura do seu próprio banco de dados SQLite. Esses exemplos são simplificados para ilustrar os conceitos básicos.

Prático:

- Instalação do SQLite;
- Execução dos primeiros comandos: .help, .databases, .tables, .schema, .pragma;
- Criação da primeira tabela (DDL);
- Inserindo, atualizando e excluindo registros (DML);
- Consultando registros (DQL);