INSTITUTO FEDERAL
SÃO PAULO
Câmpus Campos do Jordão

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SÃO PAULO

BD2A4 - BANCO DE DADOS

Professor: Paulo

ESTUDO DE CASO UTILIZANDO MongoDB

Wedyner Rodrigo Maciel – CJ3019462

Campos do Jordão

2024

RESUMO

Nesta apresentação, será explorado o universo dos bancos de dados não relacionais, ou NoSQL, destacando sua importância na gestão de grandes volumes de dados em ambientes modernos. Destacando os principais modelos NoSQL, como os bancos de dados de documentos, chave-valor, colunas, grafos, orientados a objetos e de série temporal, exemplificando seus usos e vantagens em diversas aplicações. Dentre esses modelos, terá ênfase ao MongoDB, ressaltando suas características de flexibilidade, escalabilidade e alta performance, além de discutir seus casos de uso em aplicações web, Big Data, e-commerce e redes sociais.

Figuras

INSTITUTO FEDERAL
SÃO PAULO
Câmpus Campos do Jordão

Sumário

1. INTRODUÇÃO

Com a explosão de dados gerados pela era digital, as empresas e organizações enfrentam desafios significativos na gestão, armazenamento e análise de grandes volumes de informações. Tradicionalmente, os bancos de dados relacionais (RDBMS) têm sido a escolha padrão para armazenar dados, utilizando uma estrutura de tabelas, linhas e colunas bem definidas. No entanto, à medida que os requisitos de dados evoluem, especialmente com o crescimento do Big Data, Internet das Coisas (IoT) e aplicações web dinâmicas, torna-se claro que os bancos de dados relacionais apresentam certas limitações.

Os bancos de dados não relacionais, também conhecidos como NoSQL (Not Only SQL), surgiram como uma resposta a essas limitações. Desenvolvidos para oferecer maior flexibilidade e escalabilidade, os bancos de dados NoSQL são projetados para lidar eficientemente com dados não estruturados, semi-estruturados e estruturas de dados em constante mudança. Ao contrário dos bancos de dados relacionais, que seguem um esquema rígido, os bancos de dados NoSQL permitem um modelo de dados mais flexível e adaptável.

2. Evolução e Necessidade dos Bancos de Dados Não Relacionais

A necessidade de bancos de dados não relacionais tornou-se evidente com o surgimento de grandes volumes de dados gerados por novas tecnologias e aplicativos. Aplicações de redes sociais, sistemas de e-commerce, plataformas de streaming de vídeo e áudio, e dispositivos IoT geram enormes quantidades de dados que precisam ser armazenados, processados e analisados em tempo real. Os bancos de dados relacionais, apesar de robustos, encontram dificuldades para escalar horizontalmente (adicionando mais servidores) e gerenciar dados com esquemas dinâmicos e variados.

Além disso, a natureza dos dados também mudou. Hoje, os dados não são apenas transacionais, mas também incluem textos, imagens, vídeos, documentos JSON, logs de eventos e dados de sensores. Esta diversidade de dados exige um sistema de gerenciamento de banco de dados que possa lidar com diferentes tipos de dados de maneira eficiente e flexível.

3. Características dos Bancos de Dados Não Relacionais

Os bancos de dados NoSQL são caracterizados por várias propriedades que os tornam adequados para modernas necessidades de armazenamento e processamento de dados:

Escalabilidade Horizontal: A capacidade de adicionar mais servidores para distribuir a carga de trabalho e armazenar mais dados, o que é crucial para lidar com o crescimento exponencial dos dados.

Flexibilidade na Modelagem de Dados: Permitem a inclusão de novos tipos de dados e a modificação da estrutura dos dados sem a necessidade de esquemas rígidos, tornando-os ideais para dados dinâmicos e em constante evolução.

Alto Desempenho: Projetados para otimizar operações de leitura e escrita em grandes volumes de dados, proporcionando respostas rápidas mesmo sob cargas intensas.

Disponibilidade e Particionamento: Muitos bancos de dados NoSQL são projetados para alta disponibilidade, distribuindo dados em várias máquinas para garantir que o sistema continue funcionando mesmo em caso de falhas.

4. Modelos de Bancos de Dados Não Relacionais

Os bancos de dados não relacionais, ou NoSQL, podem ser classificados em diferentes modelos, cada um projetado para atender a necessidades específicas de armazenamento e processamento de dados. A seguir, uma descrição detalhada dos principais modelos de bancos de dados não relacionais:

4.1 Bancos de Dados de Documentos (Document Stores)

Os bancos de dados de documentos armazenam dados em documentos, que são coleções auto-descritivas de pares chave-valor. Esses documentos podem ser armazenados em formatos como JSON (JavaScript Object Notation), BSON (Binary JSON) ou XML. Cada documento é uma unidade independente que pode ter uma estrutura diferente dos outros documentos, permitindo flexibilidade na modelagem de dados.

Características

- Flexibilidade na Estrutura de Dados: Cada documento pode ter uma estrutura diferente, adaptando-se facilmente a mudanças nos requisitos de dados.
- Indexação e Consultas: Suporte para criação de índices em campos dentro de documentos, permitindo consultas rápidas e eficientes.
- Escalabilidade Horizontal: Facilmente escalável adicionando mais nós ao cluster.

Exemplos:

- MongoDB: Utiliza BSON para armazenar documentos, oferecendo escalabilidade horizontal e uma linguagem de consulta poderosa. Suporta replicação e sharding para alta disponibilidade e escalabilidade.
- CouchDB: Utiliza JSON para documentos e JavaScript para consultas MapReduce. É conhecido pela sua capacidade de replicação e sincronização, permitindo a criação de aplicativos offline-first.

Casos de Uso:

- Aplicações Web Dinâmicas: Onde a estrutura dos dados pode mudar com frequência e rapidamente.
- Sistemas de Gestão de Conteúdo (CMS): Que necessitam de flexibilidade para armazenar diferentes tipos de conteúdo.
- E-commerce: Para armazenar informações de produtos com atributos variados e frequentemente mutáveis.

4.2 Bancos de Dados de Chave-Valor (Key-Value Stores)

Os bancos de dados de chave-valor armazenam dados como pares de chave-valor, onde cada chave é única e associada a um valor. O valor pode ser um dado simples ou uma estrutura complexa, mas o banco de dados trata o valor como uma string binária, sem interpretação.

Características:

- Simplicidade: Estrutura de dados extremamente simples, ideal para armazenamento rápido e eficiente.
- Alta Performance: Excelente performance para operações de leitura e escrita devido à simplicidade da estrutura de dados.
- Escalabilidade: Facilmente escalável adicionando mais nós para distribuir os pares de chave-valor.

Exemplos:

- Redis: Um armazenamento em memória que suporta várias estruturas de dados (strings, listas, conjuntos, hashes). É conhecido pela sua baixa latência e alta performance.
- DynamoDB: Um serviço gerenciado da Amazon que oferece alta disponibilidade e escalabilidade automática. Utiliza um modelo de dados de chave-valor e suporte a documentos.

Casos de Uso:

- Caches de Alta Velocidade: Para melhorar a performance de aplicativos, armazenando dados frequentemente acessados.
- Sessões de Usuário: Em aplicações web, armazenando informações temporárias sobre o usuário.
- Armazenamento de Configurações: E preferências de usuários, onde a leitura e escrita rápidas são essenciais.

4.3 Bancos de Dados de Colunas (Column-Family Stores)

Os bancos de dados de colunas armazenam dados em tabelas, mas ao contrário dos bancos de dados relacionais, as colunas são agrupadas em famílias de colunas. Cada família de colunas contém um conjunto de colunas relacionadas e permite armazenar e consultar dados de forma eficiente e escalável.

Características:

- Armazenamento Orientado a Colunas: Permite armazenar grandes volumes de dados e realizar consultas analíticas rápidas.
- Escalabilidade Horizontal: Facilmente escalável distribuindo dados em vários nós.
- Eficiência em Grandes Volumes de Dados: Ideal para operações de leitura e escrita em grandes volumes de dados.

Exemplos:

- Apache Cassandra: Oferece alta disponibilidade sem ponto único de falha, ideal para grandes volumes de dados distribuídos. Suporta replicação e partição de dados para escalabilidade.
- HBase: Baseado no Bigtable do Google, é um armazenamento distribuído para grandes tabelas de dados esparsos. Integrado com o Hadoop para processamento de grandes volumes de dados.

Casos de Uso:

- Processamento de Grandes Volumes de Dados (Big Data): Em análises de dados e sistemas de recomendação.
- Sistemas de Recomendação: Que requerem consultas rápidas em grandes conjuntos de dados.
- Aplicações de Análise de Dados em Tempo Real: Onde é necessária uma leitura rápida e eficiente de grandes volumes de dados.

4.4 Bancos de Dados de Grafos (Graph Databases)

Os bancos de dados de grafos utilizam grafos para armazenar dados, com vértices (nodos) representando entidades e arestas (arestas) representando as relações entre elas. São ideais para representar e consultar redes complexas de interconexões.

Características:

- Modelagem Natural de Relacionamentos Complexos: Permitem representar dados com múltiplas relações de maneira intuitiva.
- Consultas Eficientes: Otimizados para consultas sobre relações e conexões complexas.
- Suporte para Grafos Dirigidos e Não Dirigidos: Flexibilidade para diferentes tipos de redes.

Exemplos:

- Neo4j: Um banco de dados de grafos popular que utiliza um modelo de grafo de propriedade. Conhecido por sua performance em consultas complexas e suporte a ACID.
- ArangoDB: Um banco de dados multi-modelo que suporta documentos, grafos e chave-valor. Oferece flexibilidade na modelagem de dados e escalabilidade.

Casos de Uso

- Redes Sociais: Para modelar e analisar conexões entre usuários.
- Sistemas de Recomendação: Baseados em análise de relações e interações.
- Detecção de Fraudes: Onde é essencial identificar padrões e conexões suspeitas.

4.5 Bancos de Dados Orientados a Objetos (Object-Oriented Databases)

Os bancos de dados orientados a objetos armazenam dados na forma de objetos, como na programação orientada a objetos. Permitem que os objetos sejam persistidos diretamente no banco de dados, mantendo suas características e comportamentos.

Características:

- Integração com Linguagens de Programação Orientadas a Objetos: Permite persistir objetos diretamente do código.
- Suporte para Herança, Encapsulamento e Polimorfismo: Mantendo as características da programação orientada a objetos.
- Armazenamento de Estruturas de Dados Complexas: Adequado para aplicações com objetos complexos.

Exemplos:

- db4o: Um banco de dados orientado a objetos para Java e .NET. Facilita a integração com aplicações orientadas a objetos.
- ObjectDB: Suporta JPA e é usado principalmente em ambientes Java, oferecendo uma solução robusta para persistência de objetos.

Casos de Uso:

- Aplicações com Estruturas de Dados Complexas: Como sistemas CAD/CAM.
- Sistemas de Engenharia: Que requerem armazenamento de objetos complexos com múltiplas relações.
- Jogos e Simulações: Onde os objetos e suas interações são complexos e dinâmicos.

4.6 Bancos de Dados de Séries Temporal (Time-Series Databases)

Os bancos de dados de série temporal são otimizados para armazenar e consultar dados que são registrados ao longo do tempo, como métricas, eventos e logs. Cada ponto de dados é associado a um timestamp, permitindo consultas e análises temporais eficientes.

Características:

- Otimização para Séries Temporais: Projetados para leitura e escrita eficiente de dados temporais.
- Suporte para Compressão e Agregações: Para armazenar grandes volumes de dados de forma eficiente.
- Consultas Temporais Eficientes: Capacidade de realizar consultas complexas baseadas em tempo.

Exemplos:

- InfluxDB: Projetado especificamente para armazenamento e análise de séries temporais, com uma linguagem de consulta similar ao SQL. Suporta alta ingestão de dados e consultas rápidas.
- TimescaleDB: Uma extensão do PostgreSQL que oferece funcionalidades para séries temporais, combinando a robustez do PostgreSQL com otimizações para dados temporais.

Casos de Uso:

- Monitoramento de Infraestrutura de TI: Como servidores e redes.
- Análise de Dados Financeiros: Onde é crucial analisar dados de mercado ao longo do tempo.
- Armazenamento de Dados de Sensores IoT: Para coletar e analisar dados de dispositivos conectados em tempo real.

5. MongoDB: Um Banco de Dados de Documentos

MongoDB é um banco de dados NoSQL orientado a documentos, desenvolvido para ser escalável e de alto desempenho, oferecendo uma abordagem flexível à modelagem de dados. É um dos bancos de dados NoSQL mais populares e amplamente utilizados, especialmente em aplicações web modernas e em ambientes de Big Data.

Características do MongoDB

Modelo de Dados Flexível:

- Documentos: O MongoDB armazena dados em documentos BSON (Binary JSON). Cada documento é um conjunto de pares chave-valor, permitindo que diferentes documentos na mesma coleção tenham esquemas diferentes.
- Coleções: Conjuntos de documentos. As coleções são equivalentes às tabelas em bancos de dados relacionais, mas sem um esquema fixo.

Escalabilidade:

- Sharding: Permite distribuir dados por vários servidores, aumentando a capacidade de armazenamento e processamento. O MongoDB suporta sharding automático, o que facilita a escalabilidade horizontal.
- Replicação: Oferece replicação para alta disponibilidade e recuperação de desastres. As réplicas podem ser configuradas em clusters para garantir a redundância e a continuidade do serviço.

Performance:

- Indexação: Suporta a criação de índices em qualquer campo de um documento para melhorar a performance de consultas. Além dos índices padrão, MongoDB também oferece índices geoespaciais, compostos, textuais, entre outros.
- Consulta: A linguagem de consulta do MongoDB é rica e poderosa, permitindo operações complexas, como filtros, projeções, agregações e joins simples.

Alta Disponibilidade:

- Replica Sets: MongoDB utiliza replica sets para garantir alta disponibilidade. Um replica set consiste de múltiplas cópias dos dados, proporcionando redundância e failover automático em caso de falhas.

Agregação:

- Framework de Agregação: Oferece um framework de agregação robusto para processar dados em várias etapas, similar aos pipelines do Unix. Permite realizar operações como agrupamento, filtragem e transformação de dados.
- MapReduce: Além do framework de agregação, MongoDB suporta MapReduce para operações de agregação complexas e personalizadas.

Transações:

- Suporte a Transações ACID: Desde a versão 4.0, MongoDB oferece suporte a transações ACID multi-documento, garantindo a integridade e a consistência dos dados.

Facilidade de Uso:

- Drivers e Bibliotecas: MongoDB tem suporte para várias linguagens de programação, incluindo JavaScript (Node.js), Python, Java, C#, PHP, e muitas outras, facilitando a integração com diferentes stacks tecnológicos.
- Ferramentas de Administração: Ferramentas como MongoDB Compass oferecem interfaces gráficas para gerenciar, visualizar e consultar dados.

Casos de Uso Comuns

1. Aplicações Web Dinâmicas:

- MongoDB é amplamente utilizado em aplicações web, onde a flexibilidade do esquema permite uma rápida iteração e evolução dos dados sem interrupções.

2. Big Data e Análise de Dados:

- Graças à sua capacidade de escalar horizontalmente, MongoDB é uma escolha popular para armazenar e processar grandes volumes de dados.

3. E-commerce:

- Em sistemas de comércio eletrônico, MongoDB é utilizado para armazenar informações de produtos, catálogos, carrinhos de compras e transações.

4. Gerenciamento de Conteúdo:

- Sistemas de gerenciamento de conteúdo (CMS) aproveitam a flexibilidade do MongoDB para armazenar diferentes tipos de conteúdo e metadados.

5. Redes Sociais e Plataformas de Comunicação:

- MongoDB é adequado para armazenar perfis de usuários, posts, mensagens e outras formas de interação social devido à sua capacidade de gerenciar dados semi-estruturados.

Vantagens e Desvantagens

Vantagens:

- Flexibilidade de Esquema: Permite armazenar dados heterogêneos sem um esquema fixo, facilitando mudanças e iterações rápidas.
- Escalabilidade: Fácil de escalar horizontalmente através de sharding, suportando grandes volumes de dados e altas taxas de throughput.
- Alta Disponibilidade: Replica sets garantem a alta disponibilidade e a recuperação de desastres.
- Performance: Capacidade de indexação e consultas rápidas otimizadas para leituras e escritas de alto desempenho.

Desvantagens:

- Consistência Eventual: Em sistemas distribuídos, pode haver um pequeno atraso para que todas as réplicas fiquem consistentes.
- Complexidade na Configuração de Sharding e Replicação: A configuração de sharding e replicação pode ser complexa e requerer um gerenciamento cuidadoso.
- Limitações de Transações: Embora suporte transações ACID multi-documento, o uso intensivo de transações pode impactar a performance.

5.1 Estrutura das Coleções no MongoDB para um Estande de Tiro

Coleção de Armas

Cada documento na coleção de armas conterá informações sobre o tipo de arma, modelo, calibre, disponibilidade e número de série.

Figura 1- Coleção de Armas

Coleção de Clientes

Cada documento na coleção de clientes incluirá o nome do cliente, data de nascimento, endereço, telefone e um histórico de reservas e sessões de tiro.

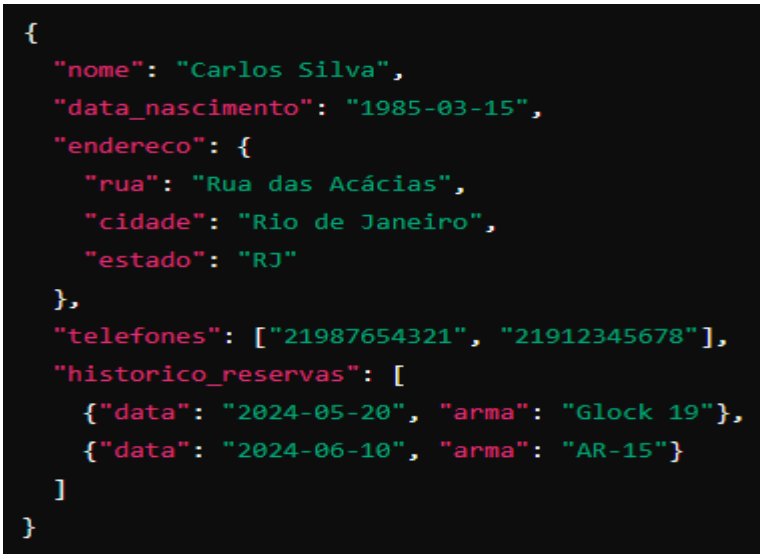

Figura 2 - Coleção de Clientes

Coleção de Sessões de Tiro

Cada documento na coleção de sessões de tiro registrará o cliente, a arma utilizada, a data da sessão, a duração e as observações.

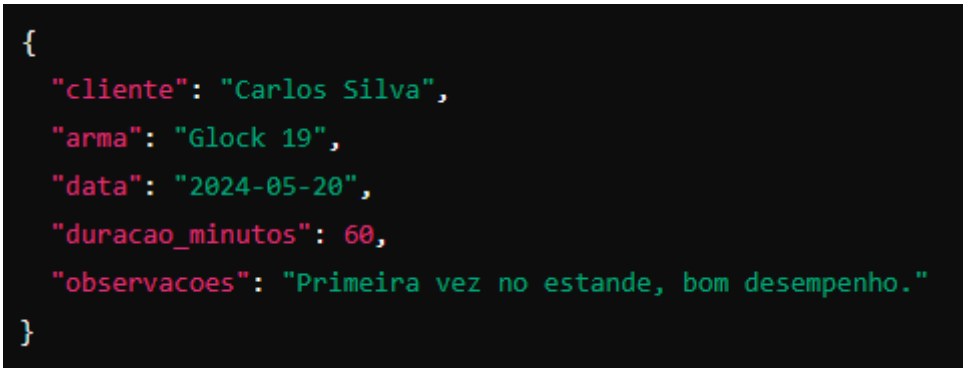

Figura 3 - Coleção de Sessões de Tiro

Coleção de Reservas

Cada documento na coleção de reservas registrará o cliente, a arma reservada, a data e o horário da reserva.

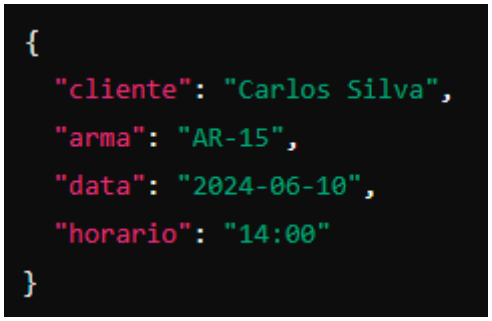

Figura 4 - Coleção de Reservas

Implementação do Projeto em C++

A implementação do projeto envolve a criação das coleções no MongoDB e a inserção de documentos conforme necessário. Utilizei o driver do MongoDB para C++, chamado `mongo`, para interagir com o banco de dados.

Conexão e Criação de Coleções

Primeiro, conectamos ao MongoDB e criamos as coleções necessárias.

```cpp
#include <bsoncxx/json.hpp>
#include <bsoncxx/builder/stream/document.hpp>
#include <mongocxx/client.hpp>
#include <mongocxx/instance.hpp>
#include <mongocxx/uri.hpp>
#include <iostream>

int main() {
    // Inicializar a instância MongoDB
    mongocxx::instance inst{};

    // Conectar ao MongoDB
    mongocxx::uri uri("mongodb://localhost:27017");
    mongocxx::client client(uri);

    // Acessar o banco de dados 'estande_de_tiro'
    mongocxx::database db = client["estande_de_tiro"];
```

```cpp
    // Acessar o banco de dados 'estande_de_tiro'
    mongocxx::database db = client["estande_de_tiro"];

    // Acessar as coleções
    mongocxx::collection armas = db["armas"];
    mongocxx::collection clientes = db["clientes"];
    mongocxx::collection sessoes = db["sessoes"];
    mongocxx::collection reservas = db["reservas"];

    return 0;
}
```

Figura 5 - Implementação do Projeto em C++

Inserção de Documentos

Inseri documentos nas coleções criadas.

```
#include <bsoncxx/json.hpp>
#include <bsoncxx/builder/stream/document.hpp>
#include <mongocxx/client.hpp>
#include <mongocxx/instance.hpp>
#include <mongocxx/uri.hpp>
#include <iostream>

int main() {
    // Inicializar a instância MongoDB
    mongocxx::instance inst{};

    // Conectar ao MongoDB
    mongocxx::uri uri("mongodb://localhost:27017");
    mongocxx::client client(uri);
```

```
    // Acessar o banco de dados 'estande_de_tiro'
    mongocxx::database db = client["estande_de_tiro"];

    // Acessar as coleções
    mongocxx::collection armas = db["armas"];
    mongocxx::collection clientes = db["clientes"];
    mongocxx::collection sessoes = db["sessoes"];
    mongocxx::collection reservas = db["reservas"];

    // Inserir documentos na coleção de armas
    bsoncxx::builder::stream::document document{};
    document << "tipo" << "Pistola"
             << "modelo" << "Glock 19"
             << "calibre" << "9mm"
             << "numero_serie" << "ABC123456"
             << "disponibilidade" << true;
```

```
armas.insert_one(document.view());

// Inserir documentos na coleção de clientes
bsoncxx::builder::stream::document doc_cliente{};
doc_cliente << "nome" << "Carlos Silva"
            << "data_nascimento" << "1985-03-15"
            << "endereco" << bsoncxx::builder::stream::open_document
            << "rua" << "Rua das Acácias"
            << "cidade" << "Rio de Janeiro"
            << "estado" << "RJ"
            << bsoncxx::builder::stream::close_document
            << "telefones" << bsoncxx::builder::stream::open_array
            << "21987654321" << "21912345678"
            << bsoncxx::builder::stream::close_array
            << "historico_reservas" << bsoncxx::builder::stream::open_array
            << bsoncxx::builder::stream::open_document
            << "data" << "2024-05-20"
```

```
            << "arma" << "Glock 19"
            << bsoncxx::builder::stream::close_document
            << bsoncxx::builder::stream::open_document
            << "data" << "2024-06-10"
            << "arma" << "AR-15"
            << bsoncxx::builder::stream::close_document
            << bsoncxx::builder::stream::close_array;

clientes.insert_one(doc_cliente.view());

// Inserir documentos na coleção de sessões de tiro
bsoncxx::builder::stream::document doc_sessao{};
doc_sessao << "cliente" << "Carlos Silva"
           << "arma" << "Glock 19"
           << "data" << "2024-05-20"
           << "duracao_minutos" << 60
```

```
            << "duracao_minutos" << 60
            << "observacoes" << "Primeira vez no estande, bom desempenho.";

    sessoes.insert_one(doc_sessao.view());

    // Inserir documentos na coleção de reservas
    bsoncxx::builder::stream::document doc_reserva{};
    doc_reserva << "cliente" << "Carlos Silva"
                << "arma" << "AR-15"
                << "data" << "2024-06-10"
                << "horario" << "14:00";

    reservas.insert_one(doc_reserva.view());

    return 0;
}
```

Figura 6 - Inserção de Documentos

Consultas e Atualizações

Podemos realizar consultas e atualizações nos documentos de forma eficiente.

```
#include <bsoncxx/json.hpp>
#include <bsoncxx/builder/stream/document.hpp>
#include <mongocxx/client.hpp>
#include <mongocxx/instance.hpp>
#include <mongocxx/uri.hpp>
#include <iostream>

int main() {
    // Inicializar a instância MongoDB
    mongocxx::instance inst{};

    // Conectar ao MongoDB
    mongocxx::uri uri("mongodb://localhost:27017");
    mongocxx::client client(uri);

    // Acessar o banco de dados 'estande_de_tiro'
    mongocxx::database db = client["estande_de_tiro"];
```

```
    // Acessar a coleção de armas
    mongocxx::collection armas = db["armas"];

    // Consultar armas disponíveis
    auto cursor = armas.find(bsoncxx::builder::stream::document{} << "disponibilidade" <<
                                                                    true << bsoncxx::builder::stream::finalize);

    for (auto&& doc : cursor) {
        std::cout << bsoncxx::to_json(doc) << std::endl;
    }

    return 0;
}
```

Figura 7 - Consultas e Atualizações

Atualização da Disponibilidade de uma Arma

```
#include <bsoncxx/json.hpp>
#include <bsoncxx/builder/stream/document.hpp>
#include <mongocxx/client.hpp>
#include <mongocxx/instance.hpp>
#include <mongocxx/uri.hpp>
#include <iostream>

int main() {
    // Inicializar a instância MongoDB
    mongocxx::instance inst{};

    // Conectar ao MongoDB
    mongocxx::uri uri("mongodb://localhost:27017");
    mongocxx::client client(uri);

    // Acessar o banco de dados 'estande_de_tiro'
    mongocxx::database db = client["estande_de_tiro"];
```

```
    // Acessar o banco de dados 'estande_de_tiro'
    mongocxx::database db = client["estande_de_tiro"];

    // Acessar a coleção de armas
    mongocxx::collection armas = db["armas"];

    // Atualizar a disponibilidade de uma arma
    bsoncxx::builder::stream::document filter_builder, update_builder;
    filter_builder << "numero_serie" << "ABC123456";
    update_builder << "$set" << bsoncxx::builder::stream::open_document << "disponibilidade" << false << bsoncxx::builder::stream::close_document;

    armas.update_one(filter_builder.view(), update_builder.view());

    return 0;
}
```

Figura 8 - Atualização da Disponibilidade de uma Arma

Consulta de Reserva de um Cliente

```
#include <bsoncxx/json.hpp>
#include <bsoncxx/builder/stream/document.hpp>
#include <mongocxx/client.hpp>
#include <mongocxx/instance.hpp>
#include <mongocxx/uri.hpp>
#include <iostream>

int main() {
    // Inicializar a instância MongoDB
    mongocxx::instance inst{};

    // Conectar ao MongoDB
    mongocxx::uri uri("mongodb://localhost:27017");
    mongocxx::client client(uri);

    // Acessar o banco de dados 'estande_de_tiro'
    mongocxx::database db = client["estande_de_tiro"];
```

```
    // Acessar a coleção de reservas
    mongocxx::collection reservas = db["reservas"];

    // Consultar reservas de um cliente
    auto cursor = reservas.find(bsoncxx::builder::stream::document{} << "cliente" << "Carlos Silva" << bsoncxx::builder::stream::finalize);

    for (auto&& doc : cursor) {
        std::cout << bsoncxx::to_json(doc) << std::endl;
    }

    return 0;
}
```

Figura 9 - Consulta de Reserva de um Cliente

6. Conclusão

Os bancos de dados não-relacionais (NoSQL) se destacam pela capacidade de lidar com grandes volumes de dados e pela flexibilidade que oferecem na modelagem das informações. Ao contrário dos bancos de dados relacionais, que utilizam tabelas rígidas com esquemas predefinidos, os bancos de dados NoSQL permitem armazenar dados em formatos variados e com estruturas que podem evoluir ao longo do tempo sem a necessidade de redefinições complexas do esquema.

Os bancos de dados NoSQL oferecem diversas vantagens, como a flexibilidade de esquema, que permite adicionar novos campos a qualquer momento sem afetar os dados existentes, o que é particularmente útil em cenários onde os requisitos do sistema evoluem rapidamente. Além disso, eles são projetados para escalar horizontalmente, adicionando mais servidores para distribuir a carga de trabalho, ideal para aplicações que precisam lidar com um crescimento exponencial de dados e acessos simultâneos. A estrutura interna dos bancos de dados NoSQL também é otimizada para oferecer alto desempenho em leituras e escritas, com mecanismos que garantem alta disponibilidade e resiliência do sistema.

O MongoDB é um exemplo de banco de dados NoSQL orientado a documentos, que utiliza documentos JSON (ou BSON) para armazenar dados. Suas principais características incluem documentos flexíveis, que permitem armazenar dados heterogêneos, suporte a consultas complexas, agregações e indexação eficiente, além de escalabilidade horizontal através de particionamento (sharding) e replicação para garantir alta disponibilidade.

Para ilustrar a aplicação prática do MongoDB, consideramos o exemplo de um sistema de gerenciamento para um estande de tiro. Nesse sistema, diferentes coleções são criadas para armazenar informações sobre armas, clientes, sessões de tiro e reservas. Cada coleção tem uma estrutura específica que reflete a natureza dos dados que armazena. A coleção de armas armazena informações detalhadas sobre cada arma disponível no estande, a coleção de clientes contém dados pessoais e históricos de reservas e sessões, a coleção de sessões de tiro registra cada sessão realizada com detalhes sobre o cliente e a arma utilizada, e a coleção de reservas mantém registros das reservas feitas pelos clientes.

Usando a linguagem C++ e a biblioteca `mongo-cxx-driver`, é possível criar um sistema que interage com o MongoDB para gerenciar o estande de tiro. A implementação envolve configurar a conexão com o servidor MongoDB, acessando o banco de dados e as coleções necessárias, inserindo novos registros nas coleções, realizando consultas para obter informações e atualizações para modificar dados.

Os benefícios do uso do MongoDB para este projeto incluem a adaptabilidade a mudanças, permitindo que novos atributos sejam adicionados aos documentos conforme necessário, sem a necessidade de migrações complexas; desempenho adequado para altas taxas de transações, essencial para um estande de tiro com alta rotatividade de clientes e sessões; escalabilidade, com a possibilidade de adicionar mais servidores para lidar com o aumento da demanda; e a capacidade de realizar consultas complexas e suportar índices para acelerar o acesso às informações mais utilizadas. A utilização do MongoDB para o gerenciamento de um estande de tiro exemplifica como um banco de dados NoSQL pode ser vantajoso em cenários onde a flexibilidade e a escalabilidade são cruciais, fornecendo um sistema robusto, eficiente e preparado para o crescimento futuro.

Referências Bibliográficas

Fowler, M., Sadalage, P. (2012). NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Addison-Wesley.

MongoDB. (n.d.). [MongoDB Manual] Disponivel <(https://www.mongodb.com/docs/manual/)> Acessado em 26 de Mai. 2024.

MongoDB. (n.d.). [MongoDB C++ Driver] Disponivel em: <(http://mongocxx.org/mongocxx-v3/)> Acessado em 26 de Mai. 2024.

Strauch, C. (2011). NoSQL Databases. Lecture Notes, Stuttgart Media University.