



UNIVERSIDADE FEDERAL DE SERGIPE
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DAVI ARAÚJO DO NASCIMENTO
GUSTAVO HENRIQUE ARAGÃO SILVA
LUCAS EMANUEL SIQUEIRA COSTA
WEMERSON DA SILVA SOARES
WILLIAM GABRIEL YCKSON ARAÚJO BRAGA

RATOPlayer - PARTE 2.2
PROJETO LÓGICO NOSQL (MONGODB)

SÃO CRISTÓVÃO/SE

2025

DAVI ARAÚJO DO NASCIMENTO
GUSTAVO HENRIQUE ARAGÃO SILVA
LUCAS EMANUEL SIQUEIRA COSTA
WEMERSON DA SILVA SOARES
WILLIAM GABRIEL YCKSON ARAÚJO BRAGA

RATOPlayer - PARTE 2.2
PROJETO LÓGICO NOSQL (MONGODB)

Relatório apresentado à Disciplina Banco de
Dados I sob orientação do Prof. Dr. André Britto
de Carvalho

São Cristóvão/SE

2025

SUMÁRIO

1. INTRODUÇÃO	4
2. MAPEAMENTO ENTRE O MODELO RELACIONAL E O MODELO NOSQL	4
3. INTRODUÇÃO DE SCHEMA NO MONGODB	5
4. MAPEAMENTO PARA A ESTRUTURA DO MODELO NOSQL	6
5. CONCLUSÕES	21
6. REFERÊNCIAS	22

1. INTRODUÇÃO

Diante da fragmentação do mercado musical atual, o RatoPlayer surge como uma proposta de plataforma integrada, capaz de unificar streaming de músicas, agenda de eventos, venda de ingressos e uma loja de produtos oficiais em um único ambiente digital. A centralização desses serviços visa oferecer uma experiência mais fluida e completa para os usuários, ao mesmo tempo em que gera dados valiosos por meio de interações como avaliações de músicas e produtos.

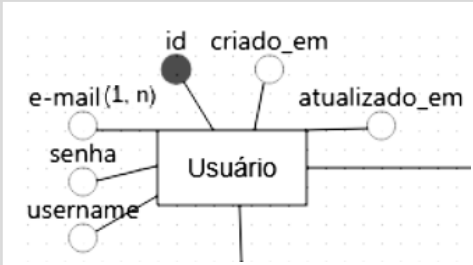
Após a definição do modelo conceitual da aplicação, com a criação do Diagrama Entidade-Relacionamento (DER), esta etapa do projeto (*Parte 2.2: Projeto Lógico NoSQL*) tem como objetivo mapear esse modelo conceitual para um sistema de banco de dados NoSQL. Isso envolve a escolha de um SGBD NoSQL apropriado, o estudo de suas características e a adaptação das entidades e relacionamentos do DER para a estrutura não relacional. O foco é garantir que as restrições e integridades originalmente propostas sejam mantidas na nova modelagem, dentro das capacidades do sistema escolhido - neste caso, o **MongoDB**.

A seguir, o mapeamento lógico da aplicação RatoPlayer no contexto NoSQL adotado.

2. MAPEAMENTO ENTRE O MODELO RELACIONAL E O MODELO NOSQL

Conforme orientado na etapa 2.2 do projeto, é necessário realizar a conversão do modelo conceitual (DER) para uma estrutura compatível com um banco de dados NoSQL, adaptando entidades e relacionamentos ao novo paradigma. Essa transição exige decisões de modelagem específicas, como a escolha entre documentos embutidos ou referenciados, o uso de arrays para representar relacionamentos com cardinalidade 1:N e a substituição de chaves primárias tradicionais por identificadores próprios do banco NoSQL.

Neste projeto, o MongoDB foi o sistema NoSQL escolhido - um banco de dados orientado a documentos, cuja estrutura se baseia em documentos BSON (semelhantes a JSON). Para ilustrar esse processo de mapeamento, a seguir é apresentada uma tabela comparativa utilizando uma das entidades principais do sistema, o usuário, como exemplo. A tabela mostra, lado a lado, como essa entidade é representada no modelo relacional (DER) e como pode ser adaptada para o formato de documento no MongoDB, destacando as principais mudanças estruturais e conceituais realizadas.

Modelo Relacional (DER)	Modelo NoSQL (Estrutura MongoDB)
	<pre>{ "_id": ObjectId("..."), "email": ["usuario@email.com", "outro@email.com"], "username": "meu_username", "senha": "hash_da_senha", "criado_em": ISODate("2025-07-22T10:00:00Z"), "atualizado_em": ISODate("2025-07-22T10:15:00Z") }</pre>

No modelo relacional, o campo **id** é utilizado como chave primária obrigatória para identificar unicamente cada registro em uma tabela. No MongoDB, essa funcionalidade é assumida automaticamente pelo campo **_id**, que é obrigatório e garante unicidade em cada documento de uma coleção. Caso o desenvolvedor não forneça um valor para o **_id**, o MongoDB gera automaticamente um identificador do tipo **ObjectId**. Dessa forma, não é necessário criar um campo **id** adicional, pois o **_id** já cumpre esse papel, sendo a chave primária implícita da estrutura de dados no modelo NoSQL. Além disso, atributos com cardinalidade (1,n), como o campo **email**, que no modelo relacional exigiriam uma tabela separada ou relacionamento, podem ser representados de forma mais simples e direta em MongoDB através de arrays. Assim, o campo **email** é modelado como um array de strings, permitindo armazenar múltiplos endereços associados a um único usuário dentro do mesmo documento, de maneira eficiente e natural para o modelo de documentos.

3. INTRODUÇÃO DE SCHEMA NO MONGODB

O MongoDB é um banco de dados NoSQL orientado a documentos, no qual os dados são armazenados em documentos BSON (semelhantes a JSON). Por padrão, o MongoDB não exige um esquema rígido, permitindo que documentos com diferentes estruturas sejam armazenados na mesma coleção. No entanto, em muitos casos, especialmente em sistemas que demandam consistência nos dados, a definição de um **schema** é recomendada para garantir que as informações armazenadas sigam padrões específicos de estrutura e validade.

A introdução de um schema no MongoDB envolve a definição de regras de validação e estruturação dos dados. Isso pode ser feito por meio de validação com **\$jsonSchema**, que permite especificar tipos de dados, campos obrigatórios e padrões de formato. A definição de um schema no MongoDB visa garantir que os dados inseridos atendam a requisitos específicos, como formato de campos, presença de dados obrigatórios e tipos de valores. A seguir, um exemplo de definição de schema para a coleção **usuarios**, baseada na estrutura previamente apresentada.

MongoDB com Schema

```
db.createCollection("usuarios", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["email", "username", "senha", "criado_em", "atualizado_em"],
      properties: {
        email: {
          bsonType: "array",
          items: {
            bsonType: "string",
            pattern: "^.+@.+\\.+\\.+ $" //valida formato de email
          }
        },
        username: {
```

```

    bsonType: "string",
    minLength: 3, //o username precisa ter pelo menos 3 caracteres
    maxLength: 30 //o username não pode ter mais de 30 caracteres
  },
  senha: {
    bsonType: "string",
    minLength: 6 //senha deve ter no mínimo 6 caracteres
  },
  criado_em: { bsonType: "date" },
  atualizado_em: { bsonType: "date" }
}
}
});

```

- **Campos obrigatórios:** O exemplo acima garante que os campos *email*, *username*, *senha*, *criado_em* e *atualizado_em* sejam obrigatórios na hora de inserir ou atualizar dados na coleção *usuarios*.
- **Validação de tipo de dado:** O MongoDB vai validar se o campo *email* é um array de strings e se os itens dentro desse array seguem o padrão de um email válido.
- **Padrão de formato:** O MongoDB pode validar o padrão de email com uma expressão regular (regex) definida no schema.
- **Validação de comprimento:** O *username* precisa ter entre 3 e 30 caracteres, e a *senha* precisa ter pelo menos 6 caracteres.

4. MAPEAMENTO PARA A ESTRUTURA DO MODELO NOSQL

A seguir, o mapeamento de todas as entidades e relacionamentos do modelo relacional definido no **Diagrama Entidade-Relacionamento (DER)** da etapa 2.1 para a estrutura NoSQL adotada, especificamente utilizando o **MongoDB**. Ao invés de múltiplas tabelas inter-relacionadas, o MongoDB trabalhará com **coleções** que representam as entidades (como *Local*, *Ouvinte*, *Amizade*, etc.), e o mapeamento refletirá as transformações necessárias para garantir a integridade dos dados e a consistência no novo modelo. Para cada entidade, será definida uma coleção com a respectiva estrutura de **schema**, abordando os tipos de dados, campos obrigatórios e validações específicas..

Local

```

db.createCollection("locais", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["nome", "estado", "pais", "logradouro", "numero"],
      properties: {
        nome: { bsonType: "string", maxLength: 45 },
        estado: { bsonType: "string", maxLength: 45 },

```

```

    pais: { bsonType: "string", maxLength: 45 },
    logradouro: { bsonType: "string", maxLength: 45 },
    numero: { bsonType: "string", maxLength: 45 }
  }
}
});

```

No MongoDB, a tabela **Local** será mapeada para uma **coleção** chamada **locais**. Ao invés de ter uma chave primária como no modelo relacional, o MongoDB usa o campo **_id** para garantir a unicidade dos documentos, o que elimina a necessidade do campo **id_local**. Os demais campos da tabela, como **nome**, **estado**, **pais**, **logradouro** e **numero**, serão diretamente representados no documento, sem necessidade de uma estrutura mais complexa ou de chaves estrangeiras.

A flexibilidade do MongoDB permite que esses dados sejam armazenados de forma simples, baseado em JSON, e o uso de validação com schema garante que os dados respeitem os tipos e a obrigatoriedade dos campos. Com isso, cada local é armazenado como um documento individual na coleção, facilitando consultas e operações sem a rigidez de um banco de dados relacional.

Ouvinte

```

db.createCollection("ouvintes", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["email", "senha", "username", "nome", "sobrenome", "data_nascimento",
"criado_em", "atualizado_em", "id_local"],
      properties: {
        email: {
          bsonType: "array",
          items: { bsonType: "string", pattern: "^.+@.+\\.+$" }, //validação de formato de email
        },
        senha: { bsonType: "string", minLength: 6 }, //senha com no mínimo 6 caracteres
        username: { bsonType: "string", minLength: 3, maxLength: 30 },
        nome: { bsonType: "string", maxLength: 30 },
        sobrenome: { bsonType: "string", maxLength: 60 },
        sexo: { bsonType: "string", enum: ["Masculino", "Feminino", "Outro"] }, //validação do tipo
enumerado
        data_nascimento: { bsonType: "date" },
        criado_em: { bsonType: "date" },
        atualizado_em: { bsonType: "date" },
        id_local: { bsonType: "objectId" } //referência para a coleção 'locais'
      }
    }
  }
});

```

No MongoDB, a tabela **Ouvinte** é mapeada para uma coleção **ouvintes**. O campo **id_usuario** se torna o **_id** gerado automaticamente, e os demais campos são armazenados diretamente no documento. A chave estrangeira **id_local** é transformada em uma referência de documento para a

coleção **locais**. O uso de **validação de schema** no MongoDB garante que os dados respeitem os tipos e regras de obrigatoriedade, assim como a unicidade do **username** e o formato dos **emails**. Além disso, com valores armazenados previamente em um campo do tipo string, ao tentar inserir um valor fora das opções válidas, o MongoDB pode rejeitar o documento, substituindo a necessidade de ENUM types.

Amizade

```
db.createCollection("amizades", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["id_seguidor", "id_seguindo"],
      properties: {
        id_seguidor: { bsonType: "objectId" },
        id_seguindo: { bsonType: "objectId" }
      }
    }
  }
});
```

A tabela **Amizade** é mapeada para a coleção **amizades**, onde a relação entre os campos *id_seguidor* e *id_seguindo* é representada como dois campos em um único documento. Ao invés de utilizar uma chave primária composta, os campos **id_seguidor** e **id_seguindo** são armazenados como referências ao **_id** dos documentos da coleção **ouvintes**. Essa abordagem elimina a necessidade de criar tabelas adicionais e facilita as consultas de relacionamento entre os usuários.

Playlist

```
db.createCollection("playlists", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["nome", "criado_em", "atualizado_em", "id_ouvinte_criador"],
      properties: {
        nome: { bsonType: "string" },
        descricao: { bsonType: "string" },
        criado_em: { bsonType: "date" },
        atualizado_em: { bsonType: "date" },
        id_ouvinte_criador: { bsonType: "objectId" }
      }
    }
  }
});
```

A tabela **Playlist** é mapeada para a coleção **playlists**, onde o campo **id_playlist** se torna o campo **_id** do documento, gerado automaticamente pelo MongoDB. A estrutura do documento inclui os campos *nome*, *descricao*, *criado_em*, *atualizado_em* e uma referência para o campo **id_ouvinte_criador**, que será representado pelo **_id** de um documento da coleção **ouvintes**. O

relacionamento entre a playlist e o ouvinte criador é armazenado como uma referência simples utilizando o **ObjectId**.

Playlists_Salvas

```
db.createCollection("playlists_salvas", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["ouvinte_id_usuario", "playlist_id_playlist"],
      properties: {
        ouvinte_id_usuario: { bsonType: "objectId" },
        playlist_id_playlist: { bsonType: "objectId" }
      }
    }
  }
});
```

A tabela **Playlists_Salvas** é mapeada para a coleção **playlists_salvas**, onde a relação entre o ouvinte e a playlist salva é representada por dois campos: *ouvinte_id_usuario* e *playlist_id_playlist*. Esses campos serão armazenados como referências para os respectivos documentos das coleções **ouvintes** e **playlists**. O uso de referências permite que o MongoDB mantenha a flexibilidade e facilite as consultas para identificar quais playlists foram salvas por determinado ouvinte.

Genero

```
db.createCollection("generos", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["nome", "surgiu_em"],
      properties: {
        nome: { bsonType: "string" },
        surgiu_em: { bsonType: "date" }
      }
    }
  }
});
```

A tabela **Genero** é mapeada para a coleção **generos**, onde o campo **nome** serve como identificador único (equivalente à chave primária) e o campo **surgiu_em** é representado como um campo do tipo **date**. Não há necessidade de relações complexas nesta tabela, pois ela contém informações simples sobre gêneros musicais. A estrutura do documento no MongoDB será direta, com a chave **nome** sendo a chave primária e a data de surgimento como um campo adicional.

Genero_Favorito

```
db.createCollection("generos_favoritos", {
```

```

validator: {
  $jsonSchema: {
    bsonType: "object",
    required: ["nome_genero", "id_usuario"],
    properties: {
      nome_genero: { bsonType: "string" },
      id_usuario: { bsonType: "objectId" }
    }
  }
}
});

```

A tabela **Genero_Favorito** relaciona um ouvinte a um gênero favorito. No MongoDB, isso será mapeado para a coleção **generos_favoritos**, onde o campo **nome_genero** será uma referência ao documento na coleção **generos** e o **id_usuario** será uma referência ao documento na coleção **ouvintes**. A combinação de **nome_genero** e **id_usuario** será utilizada como chave composta (embora o MongoDB use apenas o campo **_id**, a lógica permanece de uma relação entre as coleções). Esse relacionamento entre o ouvinte e o gênero favorito é representado por dois campos que armazenam referências.

Musica

```

db.createCollection("musicas", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["caminho_audio", "titulo"],
      properties: {
        caminho_audio: { bsonType: "string", unique: true },
        titulo: { bsonType: "string" },
        letra: { bsonType: "string" }
      }
    }
  }
});

```

A tabela **Musica** é mapeada para a coleção **musicas**, onde cada música é representada por um documento que inclui os campos *id_musica*, *letra*, *caminho_audio* e *titulo*. O campo **id_musica** será a chave primária (o campo **_id** do MongoDB) e o campo **caminho_audio** será único, já que no modelo relacional, ele deve ser único. A música pode ter uma letra associada (campo **letra**), que será armazenada como um texto.

Escutando

```

db.createCollection("escutando", {
  validator: {
    $jsonSchema: {
      bsonType: "object",

```

```

    required: ["id_usuario", "id_musica"],
    properties: {
      id_usuario: { bsonType: "objectId" },
      id_musica: { bsonType: "objectId" }
    }
  }
}
});

```

A tabela **Escutando** registra a relação entre um ouvinte e uma música que ele está ouvindo. Para o MongoDB, essa tabela será representada por uma coleção **escutando**, onde cada documento contém dois campos principais: **id_usuario** e **id_musica**, que são usados como chave composta. O relacionamento entre o ouvinte e a música será mantido de forma simples. Como o MongoDB não utiliza chave composta como no modelo relacional, os dois campos podem ser armazenados como referências para os documentos na coleção **ouvintes** e **musicas**.

Reacao

```

db.createCollection("reacoes", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["id_usuario_ouvinte", "id_musica", "id_usuario_reagiu", "emoji"],
      properties: {
        id_usuario_ouvinte: { bsonType: "objectId" },
        id_musica: { bsonType: "objectId" },
        id_usuario_reagiu: { bsonType: "objectId" },
        emoji: { bsonType: "string", minLength: 1, maxLength: 1 }
      }
    }
  }
});

```

A tabela **Reacao** representa as reações de um ouvinte sobre a música de outro ouvinte. Para a conversão para o MongoDB, isso será representado na coleção **reacoes**, onde cada documento armazena os campos *id_usuario_ouvinte*, *id_musica*, *id_usuario_reagiu* e *emoji*. A combinação dos primeiros dois campos (*id_usuario_ouvinte*, *id_musica*) representará a relação de escuta, enquanto o campo **id_usuario_reagiu** será um campo de referência ao ouvinte que reagiu à música. A chave composta de três campos do modelo relacional será convertida para referências aos documentos nas coleções **escutando** e **ouvintes**, com o campo **emoji** armazenando o símbolo da reação.

Colecao

```

db.createCollection("colecoes", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["caminho_capa", "duracao", "data_lancamento", "titulo", "tipo"],

```

```

properties: {
  caminho_capa: { bsonType: "string", unique: true },
  duracao: { bsonType: "int" },
  data_lancamento: { bsonType: "date" },
  titulo: { bsonType: "string" },
  tipo: { bsonType: "string" }
}
}
}
});

```

A tabela **Colecao** armazena informações sobre uma coleção de músicas, incluindo capa, duração, data de lançamento e tipo. No MongoDB, isso será representado como uma coleção **colecões**, com os campos **id_colecao** sendo o campo **_id** gerado automaticamente. Os campos *caminho_capa*, *duracao*, *data_lancamento*, *titulo* e *tipo* serão armazenados diretamente como campos do documento. A chave única será aplicada ao campo **caminho_capa** para garantir que a capa de cada coleção seja única. Não há necessidade de relações externas, pois essa tabela armazena apenas dados simples sobre a coleção.

Colecoes_Favoritas

```

db.createCollection("colecões_favoritas", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["id_usuario", "id_colecao"],
      properties: {
        id_usuario: { bsonType: "objectId" },
        id_colecao: { bsonType: "objectId" }
      }
    }
  }
});

```

A tabela **Colecoes_Favoritas** representa o relacionamento de um ouvinte com suas coleções favoritas. Para o MongoDB, será criada uma coleção **colecões_favoritas** onde cada documento armazena os campos **id_usuario** e **id_colecao**, com cada um referenciando os documentos nas coleções **ouvintes** e **colecões**. Em vez de usar uma chave primária composta, o MongoDB simplesmente armazenará os dois campos como uma lista de objetos.

Produtora

```

db.createCollection("produtoras", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["id_usuario", "email", "senha", "username", "nome_fantasia", "razao_social", "id_local"],

```

```

properties: {
  id_usuario: { bsonType: "objectId" },
  email: { bsonType: "array", items: { bsonType: "string" } },
  senha: { bsonType: "string" },
  username: { bsonType: "string" },
  nome_fantasia: { bsonType: "string" },
  razao_social: { bsonType: "string" },
  id_local: { bsonType: "objectId" }
}
}
});

```

A tabela **Produtora** armazena informações sobre as produtoras, incluindo dados do usuário (como email, senha, username) e dados específicos da produtora (nome fantasia, razão social, e o local). Para o MongoDB, a coleção será chamada **produtoras**, e cada documento terá um campo **_id** gerado automaticamente, enquanto o campo **id_usuario** se torna uma referência para o documento na coleção **ouvintes** (assumindo que o **id_usuario** seja de um ouvinte). Outros campos, como *nome_fantasia*, *razao_social* e *id_local*, também serão armazenados diretamente.

Artista_Banda

```

db.createCollection("artistas_bandas", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["id_usuario", "integrantes", "ano_formacao", "nome_artistico", "email", "senha",
        "username", "id_local"],
      properties: {
        id_usuario: { bsonType: "objectId" },
        integrantes: { bsonType: "array", items: { bsonType: "string" } },
        ano_formacao: { bsonType: "int" },
        nome_artistico: { bsonType: "string" },
        descricao: { bsonType: "string" },
        email: { bsonType: "array", items: { bsonType: "string" } },
        senha: { bsonType: "string" },
        username: { bsonType: "string" },
        id_produtoira: { bsonType: "objectId" },
        id_local: { bsonType: "objectId" }
      }
    }
  }
});

```

A tabela **Artista_Banda** armazena as informações de artistas e bandas, incluindo integrantes, dados do usuário (email, senha, etc.) e informações da produtora e local. No MongoDB, isso será representado como uma coleção **artistas_bandas**, onde cada documento armazena os campos do artista, como *nome_artistico*, *descricao*, *ano_formacao*, e a lista de *integrantes*. Além disso, o campo **id_produtoira** será uma referência para a produtora, e **id_local** para o local, ambos armazenados como objetos de referência.

Avaliacao

```
db.createCollection("avaliacoes", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["usuario_id", "colecacao_id", "nota"],
      properties: {
        id_usuario: { bsonType: "objectId" },
        id_colecao: { bsonType: "objectId" },
        titulo: { bsonType: "string", maxLength: 60 },
        nota: { bsonType: "decimal", minimum: 0, maximum: 10 },
        descricao: { bsonType: "string", maxLength: 500 }
      }
    }
  }
});
```

A tabela **Avaliacao** armazena avaliações feitas pelos ouvintes para as coleções, com um título, nota e descrição. Para o MongoDB, essa tabela pode ser representada como a coleção **avaliacoes**, onde cada documento conterá o **id_usuario**, **id_colecao**, **titulo**, **nota** e **descricao**. Os campos **id_usuario** e **id_colecao** serão referências para os documentos nas coleções **ouvintes** e **colecacoes**, respectivamente. A avaliação valida que o usuário e o produto são representados por um ObjectId, o campo nota tem uma restrição entre 0 e 10, e o comentário tem um limite de 500 caracteres.

Seguidores_Artistas

```
db.createCollection("seguidores_artistas", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["id_ouvinte", "id_artista"],
      properties: {
        id_ouvinte: { bsonType: "objectId" },
        id_artista: { bsonType: "objectId" }
      }
    }
  }
});
```

A tabela **Seguidores_Artistas** mantém o relacionamento de um ouvinte com os artistas que segue. No MongoDB, isso será representado por uma coleção chamada **seguidores_artistas**, onde cada documento possui **id_ouvinte** e **id_artista**, ambos como referências para os documentos nas coleções **ouvintes** e **artistas_bandas**, respectivamente.

Loja

```
db.createCollection("lojas", {
  validator: {
```

```

$jsonSchema: {
  bsonType: "object",
  required: ["artista_id", "nome", "local_id"],
  properties: {
    id_artista: { bsonType: "objectId" },
    nome: { bsonType: "string", maxLength: 60 },
    id_local: { bsonType: "objectId" }
  }
}
});

```

A tabela **Loja** mantém informações sobre as lojas dos artistas, incluindo o nome da loja e o local onde ela está localizada. Para o MongoDB, cada loja será representada como um documento na coleção **lojas**, com os campos *id_artista*, *nome* e *id_local*. O campo **id_artista** será uma referência para o documento na coleção **artistas_bandas** e **id_local** será uma referência para o documento na coleção **locais**. A estrutura pode ser armazenada diretamente, *loja* tem um único *artista* e um *local*.

Produto

```

db.createCollection("produtos", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["nome", "preco", "peso", "disponivel"],
      properties: {
        nome: { bsonType: "string" },
        preco: { bsonType: "decimal" },
        peso: { bsonType: "decimal" },
        disponivel: { bsonType: "bool" }
      }
    }
  }
});

```

A tabela **Produto** contém informações sobre os produtos disponíveis na loja, como nome, preço, peso e disponibilidade. No MongoDB, isso pode ser representado como uma coleção **produtos**, onde cada documento incluirá os campos *nome*, *preco*, *peso* e *disponivel*. O campo **nome** será a chave primária, garantindo a unicidade de cada produto.

Transacao

```

db.createCollection("transacoes", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["nome_produto", "id_ouvinte", "loja", "numero", "valor"],
      properties: {
        nome_produto: { bsonType: "string" },
        id_ouvinte: { bsonType: "objectId" },

```

```

    loja: { bsonType: "objectId" },
    numero: { bsonType: "int" },
    valor: { bsonType: "decimal" }
  }
}
});

```

A tabela **Transacao** registra as compras feitas pelos ouvintes, relacionando o produto comprado, o ouvinte e a loja. No MongoDB, cada transação pode ser armazenada como um documento na coleção **transacoes**. A coleção terá campos para **nome_produto** (referência para **produtos**), **id_ouvinte** (referência para **ouvintes**), **loja** (referência para **lojas**), **numero** (único) e **valor**. Como o campo **numero** é único, ele pode ser tratado como o identificador de cada transação.

Musicas_Playlist

```

db.createCollection("musicas_playlists", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["id_musica", "id_playlist"],
      properties: {
        id_musica: { bsonType: "objectId" },
        id_playlist: { bsonType: "objectId" }
      }
    }
  }
});

```

A tabela **Musicas_Playlist** armazena o relacionamento entre músicas e playlists. No MongoDB, isso pode ser modelado como uma coleção **musicas_playlists**, onde cada documento representará a associação entre um **id_musica** (referência para a coleção **musicas**) e um **id_playlist** (referência para a coleção **playlists**). Como a tabela relacional define a chave primária composta por ambos os campos, no MongoDB, essa relação será representada diretamente como um documento com esses dois campos.

Musica_Colecao

```

db.createCollection("musicas_colecoes", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["id_musica", "id_colecao"],
      properties: {
        id_musica: { bsonType: "objectId" },
        id_colecao: { bsonType: "objectId" }
      }
    }
  }
});

```



```
});
```

A tabela **Musica_Colecao** registra o relacionamento entre músicas e coleções. No MongoDB, essa relação pode ser armazenada em uma coleção **musicas_colecoes**, onde cada documento representará a associação de uma música a uma coleção, com os campos **id_musica** e **id_colecao** como referências para as coleções **musicas** e **colecões**, respectivamente.

Videoclipe

```
db.createCollection("videoclipes", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["id_musica", "titulo", "caminho_video", "ano_lancamento"],
      properties: {
        id_musica: { bsonType: "objectId" },
        descricao: { bsonType: "string" },
        titulo: { bsonType: "string" },
        caminho_video: { bsonType: "string" },
        ano_lancamento: { bsonType: "int" }
      }
    }
  }
});
```

A tabela **Videoclipe** armazena informações sobre videoclipes relacionados a músicas, incluindo a descrição, título, caminho do vídeo e ano de lançamento. No MongoDB, essa tabela pode ser representada como uma coleção **videoclipes**, onde cada documento conterá os campos **id_musica** (referência para a coleção **musicas**), *descricao*, *titulo*, *caminho_video* e *ano_lancamento*. O campo **caminho_video** será único para garantir a integridade dos dados.

Genero_Musica

```
db.createCollection("generos_musicas", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["nome_genero", "id_musica"],
      properties: {
        nome_genero: { bsonType: "string" },
        id_musica: { bsonType: "objectId" }
      }
    }
  }
});
```

A tabela **Genero_Musica** registra o relacionamento entre músicas e gêneros musicais. No MongoDB, isso pode ser armazenado na coleção **generos_musicas**, com cada documento

representando a associação de um gênero musical a uma música, através dos campos **nome_genero** (referência para a coleção **generos**) e **id_musica** (referência para a coleção **musicas**).

Featuring

```
db.createCollection("featuring", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["id_musica", "id_usuario"],
      properties: {
        id_musica: { bsonType: "objectId" },
        id_usuario: { bsonType: "string" }
      }
    }
  }
});
```

A tabela **Featuring** registra a colaboração de artistas ou bandas em músicas. No MongoDB, essa relação pode ser modelada na coleção **featuring**, onde cada documento representa uma colaboração entre uma música e um artista/banda. Os campos de cada documento seriam **id_musica** (referência à coleção **musicas**) e **id_usuario** (referência à coleção **artista_banda**). A relação entre os dois é representada pelo par **id_musica** e **id_usuario**.

Artista_Banda_Colecao

```
db.createCollection("artista_banda_colecao", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["id_colecao", "id_usuario"],
      properties: {
        id_colecao: { bsonType: "objectId" },
        id_usuario: { bsonType: "string" }
      }
    }
  }
});
```

A tabela **Artista_Banda_Colecao** representa a associação entre um artista/banda e as coleções de músicas. No MongoDB, a coleção **artista_banda_colecao** armazenará documentos que associam artistas/bandas às coleções. Cada documento terá **id_colecao** (referência para a coleção **colecões**) e **id_usuario** (referência para a coleção **artista_banda**), criando a relação entre o artista/banda e a coleção musical.

Genero_Colecao

```
db.createCollection("generos_colecoes", {
```

```

validator: {
  $jsonSchema: {
    bsonType: "object",
    required: ["nome_genero", "id_colecao"],
    properties: {
      nome_genero: { bsonType: "string" },
      id_colecao: { bsonType: "objectId" }
    }
  }
}
});

```

A tabela **Genero_Colecao** vincula gêneros musicais a coleções de músicas. No MongoDB, a coleção **generos_colecoes** armazenará documentos que representem a associação entre um gênero musical e uma coleção. Cada documento terá **nome_genero** (referência para a coleção **generos**) e **id_colecao** (referência para a coleção **colecões**).

Disco

```

db.createCollection("discos", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["nome_produto", "id_colecao", "tipo", "quantidade_discos"],
      properties: {
        nome_produto: { bsonType: "string" },
        id_colecao: { bsonType: "objectId" },
        tipo: { bsonType: "string" },
        quantidade_discos: { bsonType: "int" }
      }
    }
  }
});

```

A tabela **Disco** representa um produto específico, sendo um disco que pertence a uma coleção musical. No MongoDB, isso pode ser mapeado em uma coleção chamada **discos**, onde cada documento contém informações como **nome_produto** (referência à coleção **produto**), **id_colecao** (referência à coleção **colecões**), **tipo** (tipo de disco) e **quantidade_discos**.

Evento

```

db.createCollection("eventos", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["nome", "inicio", "capacidade", "id_local"],
      properties: {
        nome: { bsonType: "string" },
        inicio: { bsonType: "date" },
        capacidade: { bsonType: "int" },

```

```

    id_local: { bsonType: "objectId" }
  }
}
});

```

A tabela **Evento** descreve um evento que ocorre em um local específico. No MongoDB, a coleção **eventos** armazenará documentos com os detalhes do evento, como *nome* do evento, **inicio** (data e hora de início), **capacidade** (quantidade de pessoas permitidas) e **id_local** (referência à coleção **local**).

Atracoes

```

db.createCollection("atracoes", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["id_artista_banda", "nome_evento"],
      properties: {
        id_artista_banda: { bsonType: "string" },
        nome_evento: { bsonType: "string" }
      }
    }
  }
});

```

A tabela **Atracoes** conecta artistas/bandas a eventos. No MongoDB, isso pode ser mapeado em uma coleção chamada **atracoes**, onde cada documento representa uma atração (artista/banda) para um evento específico. O documento incluirá **id_artista_banda** (referência à coleção **artista_banda**) e **nome_evento** (referência à coleção **eventos**).

Genero_Artista_Banda

```

db.createCollection("generos_artistas_banda", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["nome_genero", "id_artista_banda"],
      properties: {
        nome_genero: { bsonType: "string" },
        id_artista_banda: { bsonType: "string" }
      }
    }
  }
});

```

A tabela **Genero_Artista_Banda** relaciona artistas/bandas com gêneros musicais, indicando quais gêneros são associados a cada artista. No MongoDB, essa tabela pode ser mapeada em uma coleção chamada **generos_artistas_banda**. Cada documento representará um artista/banda associado a um gênero, com referências tanto à coleção **genero** quanto à coleção **artista_banda**.

Vestimenta

```
db.createCollection("vestimentas", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["nome_produto", "material", "tamanho"],
      properties: {
        nome_produto: { bsonType: "string" },
        material: { bsonType: "string" },
        tamanho: { bsonType: "string" }
      }
    }
  }
});
```

A tabela **Vestimenta** descreve produtos de vestuário (como camisetas, bonés) relacionados a artistas ou bandas. No MongoDB, pode-se criar uma coleção **vestimentas**, onde cada documento representará um produto de vestuário, com campos para o nome do produto, o material e o tamanho. A referência ao produto é feita através do campo **nome_produto**, que se conecta à coleção **produto**.

Ingresso

```
db.createCollection("ingressos", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["nome_produto", "nome_evento", "tipo"],
      properties: {
        nome_produto: { bsonType: "string" },
        nome_evento: { bsonType: "string" },
        tipo: { bsonType: "string" }
      }
    }
  }
});
```

A tabela **Ingresso** relaciona ingressos a eventos específicos. Para o MongoDB, podemos criar uma coleção **ingressos**, onde cada documento representa um ingresso específico, com referências ao **produto** e ao **evento** correspondente. A tabela **Ingresso** inclui informações como o tipo de ingresso (campo **tipo**) e o nome do evento (campo **nome_evento**).

5. CONCLUSÕES

A transição do modelo relacional para o MongoDB, no contexto do projeto **RatoPlayer**, traz diversas vantagens e desafios. O MongoDB oferece flexibilidade na modelagem de dados, permitindo que informações sejam armazenadas sem a necessidade de uma estrutura rígida de tabelas, o que facilita a adaptação do sistema a mudanças rápidas nas exigências. Além disso, sua arquitetura

orientada a documentos e escalabilidade horizontal permitem que o sistema cresça de forma eficiente conforme a demanda de usuários e interações aumenta.

A modelagem de relacionamentos complexos também é simplificada, uma vez que o MongoDB utiliza arrays e documentos embutidos para representar associações, eliminando a necessidade de múltiplas tabelas. Embora o MongoDB não exija um esquema fixo, a introdução de um **schema** garante a integridade dos dados, validando tipos e formatos e assegurando consistência.

Outro ponto importante é a eliminação de chaves primárias complexas. No MongoDB, o campo **_id** já cumpre esse papel, simplificando a manipulação dos dados. A plataforma também se destaca pela alta performance, sendo ideal para sistemas que exigem escalabilidade, como o **RatoPlayer**.

No entanto, a migração para um banco NoSQL exige um novo pensamento sobre a estruturação e acesso aos dados. Além disso, desafios como a duplicação de dados ou a escolha entre documentos embutidos e referências externas precisam ser cuidadosamente avaliados, equilibrando desempenho e consistência. Em resumo, o MongoDB oferece grande flexibilidade e eficiência, mas requer atenção para garantir que a modelagem e a manipulação dos dados sejam feitas de forma estratégica.

6. REFERÊNCIAS

MONGODB, Inc. **MongoDB**. Disponível em: <https://www.mongodb.com>. Acesso em: 22 jul. 2025.

RATOPLAYER. **Diagrama de Entidades e Relacionamentos (DER)**. Disponível em: <https://i.imgur.com/o9NOSaZ>. Acesso em: 23 jul. 2025.