Universidade Federal de Sergipe
Centro de Ciências Exatas e Tecnologia
Departamento de Computação

Relatório de Banco de Dados: Fase 2.2

Projeto Lógico e Mapeamento NoSQL (Steam)

Alunos: Rafael Gomes Oliveira Santos
Alícia Vitória Sousa Santos

Professor: André Britto

São Cristóvão
2026

Sumário

1 Análise Comparativa de Modelos NoSQL

Para a seleção do SGBD, o grupo avaliou os três principais modelos não relacionais em aderência aos requisitos da plataforma Steam:

- **Chave-Valor (Ex: Redis):** Embora ofereça a menor latência possível, o modelo é limitado para consultas complexas e filtragens por múltiplos atributos (como buscar jogos por gênero e preço simultaneamente), sendo mais adequado para cache de sessões de login.
- **Família de Colunas (Ex: Cassandra):** Excelente para escrita massiva e dados distribuídos geograficamente, porém a modelagem é rígida e orientada estritamente às queries (Query-First Design). A complexidade de lidar com dados altamente aninhados (conquistas e requisitos) tornaria o desenvolvimento menos ágil.
- **Orientado a Documentos (Ex: MongoDB):** Foi o modelo escolhido por permitir o armazenamento de estruturas hierárquicas complexas em um único objeto (JSON/BSON), facilitando o mapeamento direto das entidades do DER e oferecendo um equilíbrio ideal entre flexibilidade de esquema e poder de consulta.

1.1 Justificativas Técnicas

1. **Polimorfismo de Dados:** O catálogo de produtos contém itens com atributos estruturais distintos (Jogos possuem requisitos de sistema; Softwares possuem licenças; DLCs dependem de jogos base). O modelo de documentos BSON permite armazenar essas variações na mesma coleção sem a necessidade de múltiplas tabelas e colunas nulas, comum no modelo relacional.
2. **Desempenho de Leitura (Locality of Reference):** Funcionalidades críticas, como a exibição da "Biblioteca do Usuário"ou "Detalhes do Jogo", exigem o acesso a dados agregados (usuário + jogos possuídos + conquistas). No MongoDB, esses dados podem ser modelados como documentos embutidos (*embedded documents*), permitindo sua recuperação em uma única operação de leitura, eliminando o custo computacional de *JOINs*.
3. **Escalabilidade Horizontal:** O MongoDB suporta nativamente *Sharding*, permitindo que o catálogo de produtos e o histórico de vendas cresçam indefinidamente distribuídos em clusters, atendendo ao requisito de alta disponibilidade da aplicação.

2 Mapeamento DER para NoSQL

A estratégia de mapeamento adotada foi a **Desnormalização** e o **Embedding**, reduzindo as 17 tabelas do modelo relacional para 4 coleções principais.

2.1 Estratégia de Coleções

- **Coleção `products`:** Consolida a hierarquia de herança.
  - Absorve: *Produto, Jogos, Software, DLC, RequisitoSistema, Conquista, Genero.*
  - Estratégia: Polimorfismo através de um campo discriminador `"tipo"`. Tabelas fracas como *Conquista* tornam-se arrays de subdocumentos.
- **Coleção `users`:** Focada no perfil do cliente.
  - Absorve: *Usuario, Carteira, Biblioteca.*
  - Estratégia: A tabela associativa N:N *Biblioteca* é transformada em um array dentro do documento do usuário, otimizando a consulta "Meus Jogos".
- **Coleção `orders`:** Histórico transacional imutável.
  - Absorve: *Compra, Item_compra, Nota_Fiscal.*
  - Estratégia: Padrão *Snapshot*. Os dados dos produtos (preço e título) são copiados para dentro do pedido no momento da compra para garantir histórico, independente de mudanças futuras no catálogo.
- **Coleção `reviews`:** Conteúdo gerado pelo usuário.
  - Absorve: *Avaliacao.*
  - Estratégia: Mantida como coleção separada (Referência) para evitar documentos de tamanho ilimitado (*unbounded growth*), já que um jogo popular pode ter milhões de avaliações.

3 Implementação das Restrições (Data Governance)

Embora o MongoDB seja *schema-less*, a integridade dos dados foi rigorosamente garantida utilizando **JSON Schema Validation** (*jsonSchema*). Esta funcionalidade impõe regras de estrutura no nível do banco de dados, similar ao DDL do SQL.

3.1 Validação da Coleção de Produtos

O script abaixo demonstra como garantimos a tipagem forte, campos obrigatórios e a integridade da especialização (Herança) usando o operador `oneOf`.

```
db.createCollection("products", {
   validator: {
      $jsonSchema: {
         bsonType: "object",
         required: ["tipo", "titulo", "preco", "data_lancamento"],
         properties: {
            _id: { bsonType: "objectId" },
            tipo: { enum: ["Jogo", "Software", "DLC"] }, // Enum
               Restrito
            titulo: { bsonType: "string" },
            preco: { bsonType: ["decimal", "double"], minimum: 0 },

            // Array nativo substitui tabela associativa N:N
            generos: {
               bsonType: "array",
               items: { bsonType: "string" }
            },

            // Entidade Fraca 'Conquista' embutida
            conquistas: {
               bsonType: "array",
               items: {
                  bsonType: "object",
                  required: ["titulo", "imagem"],
                  properties: {
                     titulo: { bsonType: "string" },
                     imagem: { bsonType: "binData" }
                  }
               }
            }
         },
         // Regras Condicionais de Heranca
         oneOf: [
            {
               properties: { tipo: { const: "DLC" } },
```

```
                required: ["especificacoes_dlc"] // DLC exige campo
                    especifico
            },
            {
                properties: { tipo: { const: "Jogo" } },
                required: ["detalhes_jogo"] // Jogo exige campo
                    especifico
            }
        ]
    }
  }
});
```

Listing 1: Script de Validação para Products (Polimorfismo)

3.2 Validação da Coleção de Usuários e Biblioteca

Garantia da integridade 1:1 (Carteira) e 1:N (Biblioteca) através de objetos embutidos obrigatórios.

```
db.createCollection("users", {
   validator: {
      $jsonSchema: {
         bsonType: "object",
         required: ["nome", "email", "carteira", "biblioteca"],
         properties: {
            email: {
                bsonType: "string",
                pattern: "^.+@.+$" // Regex para validacao de email
            },
            // Relacionamento 1:1 estrito (Carteira)
            carteira: {
               bsonType: "object",
               required: ["saldo", "moeda"],
               properties: {
                  saldo: { bsonType: "decimal" },
                  moeda: { bsonType: "string", minLength: 3, maxLength:
                      3 }
               }
            },
            // Relacionamento N:N transformado em 1:N (Biblioteca)
            biblioteca: {
               bsonType: "array",
               items: {
                  bsonType: "object",
                  required: ["produto_id", "data_aquisicao"],
                  properties: {
```

```
                    produto_id: { bsonType: "objectId" }, // Referencia
                    status_instalacao: { bsonType: "bool" }
                }
            }
        }
    }
}
});
```

Listing 2: Script de Validação para Users

4 Conclusão

O mapeamento para NoSQL permitiu reduzir a complexidade de junções do modelo relacional, favorecendo a performance de leitura crítica para a plataforma. As restrições de integridade, antes garantidas por Chaves Estrangeiras, foram transpostas com sucesso para Validadores de Esquema e regras de aplicação, assegurando a consistência dos dados mesmo em um ambiente flexível.

** O script completo de criação das 4 coleções encontra-se no arquivo* `criar_colecoes_steam.js` *em anexo.*