

MongoDB Shell: Notação, Sintaxe e Semântica

1. Introdução

O **MongoDB Shell** (mongosh) é um ambiente interactivo baseado em JavaScript que permite aos utilizadores interagir com instâncias MongoDB através da linha de comando. Desenvolvido pela MongoDB Inc., é a interface padrão para executar operações em bases de dados NoSQL documentais. O MongoDB Shell suporta toda a sintaxe JavaScript moderna (ECMAScript), permitindo automação, scripting e execução de operações complexas.

Este documento apresenta uma análise científica completa do MongoDB Shell, cobrindo a sua notação, sintaxe e semântica, com exemplos práticos para cada conceito apresentado. MongoDB utiliza a **MongoDB Query Language (MQL)**, que é fundamentalmente diferente do SQL relacional, operando sobre documentos JSON em vez de tabelas.

2. Conceitos Fundamentais

2.1 Terminologia MongoDB vs. SQL

Correspondência entre a terminologia de SQL e MongoDB:

Conceito SQL	Conceito MongoDB	Descrição
Base de Dados (Database)	Database	Colecção de colecções
Tabela (Table)	Colecção (Collection)	Estrutura que armazena documentos
Linha (Row/Record)	Documento (Document)	Um registo de dados em formato JSON/BSON
Coluna (Column)	Campo (Field)	Uma propriedade dentro de um documento
Índice (Index)	Índice (Index)	Estrutura que acelera consultas
Chave Primária	<code>_id</code> Field	Identificador único obrigatório

2.2 Estrutura de um Documento MongoDB

Os documentos MongoDB são estruturas JSON-like com suporte adicional de tipos BSON (Binary JSON).

Exemplo de Documento:

```
{  
  "_id": ObjectId("507f1f77bcf86cd799439011"),  
  "nome": "João Silva",  
  "idade": 28,  
  "activo": true,  
  "email": "joao@example.com",  
  "data_criacao": ISODate("2025-11-06T20:14:00Z"),  
  "departamento": {  
    "id": 1,  
    "nome": "Departamento A"  
  }  
}
```

```

        "nome": "Tecnologia da Informação"
    },
    "competencias": ["Python", "JavaScript", "MongoDB"],
    "salario": NumberDecimal("3500.50")
}

```

2.3 O Campo `_id`

Todo o documento MongoDB possui um campo `_id` único, que actua como chave primária:

Características:

- Obrigatório em todos os documentos
- Gerado automaticamente como ObjectId se não for especificado
- Pode ser qualquer tipo de dado (String, Number, ObjectId, etc.)
- Garante a unicidade de cada documento na coleção

Exemplo:

```

// ObjectId gerado automaticamente
db.usuarios.insertOne({ nome: "Maria" });
// Resultado: { "_id": ObjectId(...), "nome": "Maria" }

// ObjectId personalizado
db.usuarios.insertOne({ _id: "user_123", nome: "Carlos" });

// String como _id
db.usuarios.insertOne({ _id: "carlossilva@example.com", nome: "Carlos
Silva" });

```

3. Notação e Convenções Sintácticas

3.1 Notação BNF Estendida para MongoDB

A sintaxe do MongoDB Shell segue convenções específicas:

Notação	Significado	Exemplo
{ }	Documento/Objecto	{ nome: "João", idade: 30 }
[]	Array/Lista	["Python", "JavaScript", "SQL"]
db	Referência à base de dados actual	db.coleccao
db.coleccao	Referência a uma coleção	db.usuarios
\$	Prefixo de operador	{ \$gt: 100 }
:	Separador chave-valor	{ campo: valor }
...	Múltiplos argumentos	{ campo1: val1, campo2: val2, ... }

Notação	Significado	Exemplo
[] (opcional)	Elemento opcional	[opcao]
{ \ }	Escolha de alternativas	{ \$eq \ \$ne \ \$gt }

Exemplo de Sintaxe:

```
db.coleccao.metodo({
  { campo: valor, ... },      // Filtro/Query
  [ opcoes ]                  // Opções (opcional)
});
```

3.2 Convenções de Nomenclatura

Regras para Identificadores:

- Começar com letra (a-z, A-Z) ou underscore (_)
- Podem conter letras, dígitos, underscores
- São case-sensitive (diferenciam maiúsculas de minúsculas)
- Não podem conter espaços
- Usar convenção camelCase ou snake_case

Exemplos Corretos:

```
// Correcto
db.usuarios           // snake_case para coleções
db.departamentos      // nome significativo
db.log_auditoria     // uso de underscore

// Campos em documentos
{ nome_funcionario: "João", data_admissao: "2020-01-15", ativo: true }

// Incorrecto
db.u                  // muito genérico
db.123coleccao        // começa com número
db.meu coleccao       // contém espaço
```

4. Tipos de Dados BSON

4.1 Tipos de Dados Primitivos

4.1.1 String (UTF-8)

Cadeia de caracteres em UTF-8. Máximo de 16 MB por documento.

```
{ nome: "João Silva", email: "joao@example.com", descricao: "Engenheiro de Software" }
```

4.1.2 Integer

Números inteiros de 32 bits (Int32) ou 64 bits (Int64).

```
{
  idade: 28,                                // Int32 (padrão)
  identificador: NumberLong("9223372036854775807") // Int64
}
```

4.1.3 Double

Números em ponto flutuante (64 bits).

```
{
  altura: 1.75,
  peso: 72.5,
  desconto_percentual: 15.0
}
```

4.1.4 Decimal

Números decimais de alta precisão (128 bits), ideais para operações financeiras.

```
{
  salario: NumberDecimal("3500.50"),
  preco_produto: NumberDecimal("99.99"),
  taxa_juro: NumberDecimal("0.035")
}
```

4.1.5 Boolean

Valores lógicos verdadeiro ou falso.

```
{
  activo: true,
  verificado: false,
  premium: true
}
```

4.1.6 Null

Valor nulo, representa ausência de valor.

```
{  
  nome: "João",  
  titulo: null,           // Campo nulo  
  data_termino: null     // Campo opcional não preenchido  
}
```

4.1.7 ObjectId

Identificador único de 12 bytes. Estrutura: 4 bytes (timestamp) + 5 bytes (valor aleatório) + 3 bytes (contador).

```
{  
  _id: ObjectId("507f1f77bcf86cd799439011"),  
  usuario_id: ObjectId("507f1f77bcf86cd799439012")  
}
```

4.1.8 Date (ISODate)

Data e hora em formato ISO 8601, armazenadas como milissegundos desde epoch (1 de Janeiro de 1970).

```
{  
  data_criacao: ISODate("2025-11-06T20:14:00Z"),  
  data_atualizacao: new Date(),  
  data_termino: ISODate("2025-12-31")  
}
```

4.1.9 Binary Data

Dados binários (imagens, ficheiros, etc.). Máximo 16 MB.

```
{  
  _id: ObjectId(...),  
  foto_perfil: BinData(0, "base64encodeddata=="),  
  ficheiro_pdf: BinData(0, "...")  
}
```

4.1.10 Regular Expression

Expressões regulares para pattern matching.

```
{
  email: /^[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,}$/,
  telefone: /^\d{3}-\d{3}-\d{4}$/
}
```

4.2 Tipos de Dados Compostos

4.2.1 Document (Embedded Document)

Documentos aninhados para representar dados hierárquicos.

```
{
  _id: ObjectId(...),
  nome: "João Silva",
  endereco: {
    rua: "Rua Principal",
    numero: 123,
    localidade: "Braga",
    codigo_postal: "4700-000",
    pais: "Portugal"
  },
  contacto: {
    email: "joao@example.com",
    telefone: "+351912345678"
  }
}
```

4.2.2 Array

Coleção de valores de qualquer tipo.

```
{
  _id: ObjectId(...),
  nome: "Project Alpha",
  tags: ["urgente", "prioritario", "desenvolvimento"],
  membros: [
    { nome: "João", papel: "Lead" },
    { nome: "Maria", papel: "Developer" }
  ],
  orcamentos: [5000, 7500, 10000],
  coordenadas: [40.1916, -8.6869] // [latitude, longitude]
}
```

4.2.3 Timestamp

Timestamp especial usado internamente por MongoDB para replicação.

```
{  
  _id: ObjectId(...),  
  versao: Timestamp(1, 1) // (time, increment)  
}
```

5. Conectar à Base de Dados

5.1 Conexão Básica

Sintaxe:

```
mongosh "mongodb://[utilizador:senha@]host[:porta]/[base_dados]"
```

Exemplos:

```
// Conexão local padrão  
mongosh  
  
// Conexão com especificação de host  
mongosh "mongodb://localhost:27017"  
  
// Conexão com autenticação  
mongosh "mongodb://user:password@localhost:27017/minha_bd"  
  
// Conexão a MongoDB Atlas  
mongosh "mongodb+srv://user:password@cluster.mongodb.net/minha_bd"
```

5.2 Comandos de Navegação Básica

```
// Ver todas as bases de dados  
show databases  
  
// Seleccionar base de dados  
use minha_base_dados  
  
// Ver base de dados actual  
db  
  
// Ver todas as coleções  
show collections  
  
// Ver informações da coleção  
db.coleccao.stats()  
  
// Obter ajuda
```

```
help
db.help()
db.coleccao.help()
```

6. Operações CRUD

6.1 CREATE (Inserir Documentos)

6.1.1 insertOne()

Insere um único documento.

Sintaxe:

```
db.coleccao.insertOne(documento, [opcoes])
```

Exemplo:

```
db.usuarios.insertOne({
  nome: "João Silva",
  email: "joao@example.com",
  idade: 28,
  departamento: "TI",
  activo: true,
  data_criacao: new Date()
});

// Resultado
{
  acknowledged: true,
  insertedId: ObjectId("507f1f77bcf86cd799439011")
}
```

6.1.2 insertMany()

Insere múltiplos documentos de uma vez.

Sintaxe:

```
db.coleccao.insertMany([documento1, documento2, ...], [opcoes])
```

Opções:

- `ordered: true` (por defeito) - Inserção ordenada, para se houver erro
- `ordered: false` - Tenta inserir todos, mesmo se alguns falharem

Exemplo:

```
db.usuarios.insertMany([
  { nome: "João Silva", idade: 28, departamento: "TI" },
  { nome: "Maria Santos", idade: 32, departamento: "RH" },
  { nome: "Carlos Oliveira", idade: 25, departamento: "Vendas" }
]);

// Com opção ordered: false
db.usuarios.insertMany(
[
  { nome: "Ana", email: "ana@example.com" },
  { nome: "Bruno", email: "bruno@example.com" }
],
{ ordered: false }
);
```

6.2 READ (Consultar Documentos)

6.2.1 find()

Consulta documentos da coleção.

Sintaxe:

```
db.coleccao.find(filtro, [projecao])
```

Parâmetros:

- **filtro**: Condições para selecionar documentos (opcional, `{}` seleciona tudo)
- **projecao**: Campos a retornar (opcional)

Exemplo Básico:

```
// Todos os documentos
db.usuarios.find()

// Documentos com pretty print
db.usuarios.find().pretty()

// Primeiros 5 documentos
db.usuarios.find().limit(5)

// Mostrar apenas nome e email
db.usuarios.find({}, { nome: 1, email: 1 })
```

6.2.2 Operadores de Comparação

Operador	Descrição	Sintaxe
\$eq	Igual a	{ campo: { \$eq: valor } }
\$ne	Não igual a	{ campo: { \$ne: valor } }
\$gt	Maior que	{ campo: { \$gt: valor } }
\$gte	Maior ou igual a	{ campo: { \$gte: valor } }
\$lt	Menor que	{ campo: { \$lt: valor } }
\$lte	Menor ou igual a	{ campo: { \$lte: valor } }

Exemplo:

```
// Utilizadores com idade > 25
db.usuarios.find({ idade: { $gt: 25 } })

// Utilizadores com idade entre 25 e 35
db.usuarios.find({
  idade: { $gte: 25, $lte: 35 }
})

// Utilizadores que não são do departamento TI
db.usuarios.find({ departamento: { $ne: "TI" } })
```

6.2.3 Operadores Lógicos

Operador	Descrição	Sintaxe
\$and	Ambas as condições verdadeiras	{ \$and: [{ cond1 }, { cond2 }] }
\$or	Uma das condições verdadeira	{ \$or: [{ cond1 }, { cond2 }] }
\$nor	Nenhuma das condições verdadeira	{ \$nor: [{ cond1 }, { cond2 }] }
\$not	Nega a condição	{ campo: { \$not: { cond } } }

Exemplo:

```
// Utilizadores no departamento TI com idade > 25
db.usuarios.find({
  $and: [
    { departamento: "TI" },
    { idade: { $gt: 25 } }
  ]
})

// Utilizadores em RH OU Vendas
db.usuarios.find({
  $or: [
    { departamento: "RH" },
    { departamento: "Vendas" }
  ]
})
```

```

        { departamento: "RH" },
        { departamento: "Vendas" }
    ]
})

// Utilizadores que não estão em TI e não são activos
db.usuarios.find({
    $nor: [
        { departamento: "TI" },
        { activo: true }
    ]
})

```

6.2.4 Operadores de Array

Operador	Descrição	Sintaxe
\$in	Valor está numa lista	{ campo: { \$in: [val1, val2] } }
\$nin	Valor não está numa lista	{ campo: { \$nin: [val1, val2] } }
\$all	Array contém todos os valores	{ array: { \$all: [val1, val2] } }
\$elemMatch	Array elemento satisfaz condição	{ array: { \$elemMatch: { cond } } }

Exemplo:

```

// Utilizadores nos departamentos especificados
db.usuarios.find({
    departamento: { $in: ["TI", "RH", "Marketing"] }
})

// Utilizadores com competências específicas
db.usuarios.find({
    competencias: { $all: ["Python", "JavaScript"] }
})

// Utilizadores com idade em intervalo
db.usuarios.find({
    idade: { $in: [25, 30, 35] }
})

```

6.2.5 Operadores de String

Operador	Descrição	Sintaxe
\$regex	Pattern matching com regex	{ campo: { \$regex: padrão } }
\$text	Text search (requer index)	{ \$text: { \$search: texto } }

Exemplo:

```
// Nomes que começam com "João"
db.usuarios.find({
  nome: { $regex: "^João" }
})

// Email que contém "example"
db.usuarios.find({
  email: { $regex: "example", $options: "i" } // i = case-insensitive
})

// Text search (requer índice de texto criado)
db.usuarios.find({
  $text: { $search: "engenheiro software" }
})
```

6.2.6 Operadores de Nulidade

```
// Documentos onde o campo existe
db.usuarios.find({ titulo: { $exists: true } })

// Documentos onde o campo NÃO existe
db.usuarios.find({ titulo: { $exists: false } })

// Documentos onde o campo é null
db.usuarios.find({ titulo: null })

// Documentos onde o campo NÃO é null
db.usuarios.find({ titulo: { $ne: null } })
```

6.2.7 Métodos de Formatação

```
// Pretty print
db.usuarios.find().pretty()

// Ordenação (1 = ascendente, -1 = descendente)
db.usuarios.find().sort({ nome: 1, idade: -1 })

// Limitar número de documentos
db.usuarios.find().limit(10)

// Saltar documentos (paginação)
db.usuarios.find().skip(5).limit(10)

// Contar documentos
db.usuarios.find().count()
```

```
db.usuarios.countDocuments({ departamento: "TI" })

// Encontrar um documento
db.usuarios.findOne({ email: "joao@example.com" })
```

6.3 UPDATE (Actualizar Documentos)

6.3.1 updateOne()

Actualiza o primeiro documento que corresponde ao filtro.

Sintaxe:

```
db.coleccao.updateOne(filtro, operacao_atualizacao, [opcoes])
```

Exemplo:

```
// Actualizar o nome de um utilizador
db.usuarios.updateOne(
  { _id: ObjectId("507f1f77bcf86cd799439011") },
  { $set: { nome: "João Pedro Silva" } }
);
```

6.3.2 updateMany()

Actualiza todos os documentos que correspondem ao filtro.

Exemplo:

```
// Aumentar o salário de todos os funcionários do departamento TI em 10%
db.usuarios.updateMany(
  { departamento: "TI" },
  { $mul: { salario: 1.10 } }
);
```

6.3.3 Operadores de Actualização

Operador	Descrição	Sintaxe
\$set	Define valor do campo	{ \$set: { campo: valor } }
\$unset	Remove o campo	{ \$unset: { campo: "" } }
\$inc	Incrementa valor numérico	{ \$inc: { campo: quantidade } }
\$mul	Multiplica valor numérico	{ \$mul: { campo: factor } }

Operador	Descrição	Sintaxe
\$rename	Renomeia campo	{ \$rename: { velho: novo } }
\$push	Adiciona elemento ao array	{ \$push: { array: elemento } }
\$pull	Remove elemento do array	{ \$pull: { array: elemento } }
\$addToSet	Adiciona elemento único ao array	{ \$addToSet: { array: elemento } }
\$pop	Remove primeiro/último elemento	{ \$pop: { array: 1 ou -1 } }

Exemplos:

```
// $set - Actualizar campo
db.usuarios.updateOne(
  { _id: ObjectId(...) },
  { $set: { email: "novo@example.com", activo: true } }
);

// $inc - Incrementar contador
db.usuarios.updateOne(
  { _id: ObjectId(...) },
  { $inc: { anos_servicio: 1, numPostosProvovidos: 1 } }
);

// $unset - Remover campo
db.usuarios.updateOne(
  { _id: ObjectId(...) },
  { $unset: { titulo_temporario: "" } }
);

// $rename - Renomear campo
db.usuarios.updateMany(
  { },
  { $rename: { "endereco.pais": "endereco.pais_novo" } }
);

// $push - Adicionar valor a array
db.usuarios.updateOne(
  { _id: ObjectId(...) },
  { $push: { competencias: "Docker" } }
);

// $addToSet - Adicionar valor único ao array
db.usuarios.updateOne(
  { _id: ObjectId(...) },
  { $addToSet: { competencias: "Kubernetes" } }
);

// $pull - Remover valor do array
db.usuarios.updateOne(
  { _id: ObjectId(...) },
```

```
{ $pull: { competencias: "PHP" } }
);

// $pop - Remover primeiro (-1) ou último (1) elemento
db.usuarios.updateOne(
  { _id: ObjectId(...) },
  { $pop: { competencias: 1 } } // Remove último
);
```

6.3.4 Operação Upsert

Cria documento se não existir, senão actualiza.

```
db.usuarios.updateOne(
  { email: "novo@example.com" },
  { $set: { nome: "Novo Utilizador", departamento: "TI" } },
  { upsert: true }
);
```

6.3.5 replaceOne()

Substitui documento inteiro (excepto `_id`).

```
db.usuarios.replaceOne(
  { _id: ObjectId(...) },
  {
    nome: "João Silva",
    email: "joao@example.com",
    departamento: "TI",
    activo: true
  }
);
```

6.4 DELETE (Remover Documentos)

6.4.1 deleteOne()

Remove o primeiro documento que corresponde ao filtro.

Sintaxe:

```
db.coleccao.deleteOne(filtro, [opcoes])
```

Exemplo:

```
// Remover utilizador específico  
db.usuarios.deleteOne({ _id: ObjectId("507f1f77bcf86cd799439011") });  
  
// Remover primeiro utilizador inactivo  
db.usuarios.deleteOne({ activo: false });
```

6.4.2 deleteMany()

Remove todos os documentos que correspondem ao filtro.

Exemplo:

```
// Remover todos os utilizadores inactivos  
db.usuarios.deleteMany({ activo: false });  
  
// Remover todos os utilizadores do departamento TI  
db.usuarios.deleteMany({ departamento: "TI" });  
  
// Remover todos os documentos da colecção  
db.usuarios.deleteMany({});
```

7. Agregação

7.1 Framework de Agregação

Pipeline de múltiplos estágios para transformação de dados.

Sintaxe:

```
db.coleccao.aggregate([  
  { $stage1: { operação1 } },  
  { $stage2: { operação2 } },  
  ...  
])
```

7.2 Estágios de Agregação Principais

7.2.1 \$match

Filtre documentos baseado em critérios.

```
db.usuarios.aggregate([  
  {  
    $match: {  
      departamento: "TI",  
    }  
  }  
])
```

```

        idade: { $gt: 25 }
    }
]);

```

7.2.2 \$group

Agrupa documentos por um campo e calcula agregações.

Sintaxe:

```
{
  $group: {
    _id: valor_agrupamento,
    campo_agregado: { operador_agregacao: "campo" }
  }
}
```

Operadores de Agregação:

Operador	Descrição
\$sum	Soma valores
\$avg	Média de valores
\$min	Valor mínimo
\$max	Valor máximo
\$count	Conta documentos
\$push	Coleciona valores em array
\$first	Primeiro valor
\$last	Último valor

Exemplo:

```
// Total de funcionários e salário médio por departamento
db.usuarios.aggregate([
{
  $group: {
    _id: "$departamento",
    total_funcionarios: { $sum: 1 },
    salario_medio: { $avg: "$salario" },
    salario_maximo: { $max: "$salario" },
    salario_minimo: { $min: "$salario" }
  }
}]
```

```
    }
]);
```

7.2.3 \$project

Seleciona, calcula e transforma campos.

```
db.usuarios.aggregate([
{
  $project: {
    nome: 1,
    email: 1,
    salario_anual: { $multiply: ["$salario", 12] },
    _id: 0
  }
});
]);
```

7.2.4 \$sort

Ordena documentos.

```
db.usuarios.aggregate([
{
  $sort: { salario: -1, nome: 1 } // -1 descending, 1 ascending
}
]);
]);
```

7.2.5 \$limit e \$skip

Limita e salta documentos (paginação).

```
db.usuarios.aggregate([
  { $skip: 10 },
  { $limit: 5 }
]);
]);
```

7.2.6 \$lookup

Junta dados de outra colecção (similar a JOIN em SQL).

```
db.funcionarios.aggregate([
{
  $lookup: {
```

```
        from: "departamentos",
        localField: "id_departamento",
        foreignField: "_id",
        as: "departamento_info"
    }
]
]);
```

7.2.7 \$unwind

Desconstrói arrays em documentos separados.

```
// Se competencias é um array
db.usuarios.aggregate([
{
    $unwind: "$competencias"
}
]);
```

7.3 Pipeline Agregação Completo

```
db.usuarios.aggregate([
    // Filtrar
    {
        $match: {
            departamento: "TI",
            activo: true
        }
    },
    // Agrupar e calcular
    {
        $group: {
            _id: "$departamento",
            total: { $sum: 1 },
            salario_medio: { $avg: "$salario" },
            salario_maximo: { $max: "$salario" }
        }
    },
    // Ordenar
    {
        $sort: { salario_medio: -1 }
    },
    // Limitar
    {
        $limit: 5
    }
]);
```

8. Índices

8.1 createIndex()

Cria índice para acelerar consultas.

Sintaxe:

```
db.coleccao.createIndex(especificacao, [opcoes])
```

Especificação:

- **1** = Índice ascendente
- **-1** = Índice descendente

Exemplos:

```
// Índice simples  
db.usuarios.createIndex({ email: 1 });

// Índice composto  
db.usuarios.createIndex({ departamento: 1, nome: 1 });

// Índice único  
db.usuarios.createIndex({ email: 1 }, { unique: true });

// Índice de texto  
db.usuarios.createIndex({ descricao: "text", nome: "text" });
```

8.2 getIndexes()

Lista todos os índices.

```
db.usuarios.getIndexes();
```

8.3 dropIndex()

Remove um índice.

```
// Remover por nome  
db.usuarios.dropIndex("email_1");

// Remover por especificação  
db.usuarios.dropIndex({ email: 1 });
```

```
// Remover todos excepto _id
db.usuarios.dropIndexes();
```

9. Validação de Schema

9.1 createCollection() com Validação

Define regras de validação JSON Schema.

Sintaxe:

```
db.createCollection("coleccao", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["nome", "email"],
      properties: {
        nome: {
          bsonType: "string",
          description: "deve ser string"
        },
        email: {
          bsonType: "string",
          pattern: "^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.\.[a-zA-Z]{2,}$",
          description: "deve ser email válido"
        },
        idade: {
          bsonType: "int",
          minimum: 18,
          maximum: 120,
          description: "deve estar entre 18 e 120"
        }
      }
    }
  }
});
```

9.2 Modificar Validação

```
db.runCommand({
  collMod: "usuarios",
  validator: {
    $jsonSchema: {
      // Novo schema
    }
  }
});
```

10. Transações

10.1 Transações Multi-Dокументos

Operações ACID across múltiplos documentos.

Sintaxe:

```
session = db.getMongo().startSession();

session.startTransaction();
try {
    // Operações
    db.usuarios.updateOne(filtro1, atualizacao1, { session: session });
    db.departamentos.updateOne(filtro2, atualizacao2, { session: session });

    session.commitTransaction();
} catch (error) {
    session.abortTransaction();
    throw error;
} finally {
    session.endSession();
}
```

Exemplo Prático:

```
session = db.getMongo().startSession();

session.startTransaction();
try {
    // Transferência: remover de uma conta e adicionar a outra
    db.contas.updateOne(
        { _id: "conta_origem" },
        { $inc: { saldo: -100 } },
        { session: session }
    );

    db.contas.updateOne(
        { _id: "conta_destino" },
        { $inc: { saldo: 100 } },
        { session: session }
    );

    session.commitTransaction();
    print("Transação confirmada");
} catch (error) {
    session.abortTransaction();
    print("Transação cancelada: " + error);
} finally {
```

```
    session.endSession();
}
```

11. Operações em Lote

11.1 bulkWrite()

Executa múltiplas operações de escrita de forma eficiente.

Sintaxe:

```
db.coleccao.bulkWrite([
  { insertOne: { document: {...} } },
  { updateOne: { filter: {...}, update: {...} } },
  { deleteOne: { filter: {...} } },
  ...
])
```

Exemplo:

```
db.usuarios.bulkWrite([
  {
    insertOne: {
      document: { nome: "João", email: "joao@example.com" }
    }
  },
  {
    updateOne: {
      filter: { _id: ObjectId(...) },
      update: { $set: { activo: false } }
    }
  },
  {
    deleteMany: {
      filter: { departamento: "Obsoleto" }
    }
  }
]);
```

12. Operações Avançadas

12.1 Distinct

Retorna valores únicos de um campo.

```
// Todos os departamentos únicos
db.usuarios.distinct("departamento");
```

```
// Departamentos únicos de utilizadores activos
db.usuarios.distinct("departamento", { activo: true });
```

12.2 Text Search

Procura de texto completo.

```
// Criar índice de texto
db.usuarios.createIndex({ nome: "text", email: "text" });

// Fazer procura
db.usuarios.find({
  $text: { $search: "João Silva" }
}).projection({ score: { $meta: "textScore" } })
  .sort({ score: { $meta: "textScore" } });
```

12.3 FindAndModify

Encontra e modifica documento, retornando valores.

```
db.usuarios.findAndModify({
  query: { _id: ObjectId(...) },
  update: { $set: { status: "processando" } },
  new: true // Retorna documento modificado
});
```

13. Boas Práticas e Considerações de Semântica

13.1 Tratamento de Null vs. Campo Ausente

```
// Null explícito
{ _id: 1, nome: "João", titulo: null }

// Campo ausente
{ _id: 2, nome: "Maria" }

// Consultar null
db.usuarios.find({ titulo: null });           // Ambos acima
db.usuarios.find({ titulo: { $exists: false } }); // Apenas second

// Consultar ambos
db.usuarios.find({ titulo: { $in: [null] } });
```

13.2 Type Sensitivity

MongoDB é type-sensitive em comparações:

```
// Estes são diferentes
db.usuarios.find({ idade: 25 });           // Número
db.usuarios.find({ idade: "25" });          // String

// Comparações entre tipos
db.usuarios.find({ idade: { $gt: "20" } }); // String comparison
```

13.3 Conversão de Tipos

```
// Conversão de String para Number
db.usuarios.find({ idade: { $type: "int" } });

// Usar aggregation para converter
db.usuarios.aggregate([
  {
    $project: {
      idade_numero: { $toInt: "$idade" }
    }
  }
]);
```

13.4 Performance: Projections

Sempre utilizar projections para limitar campos retornados:

```
// Ineficiente – retorna documento inteiro
db.usuarios.find({ departamento: "TI" });

// Eficiente – apenas campos necessários
db.usuarios.find(
  { departamento: "TI" },
  { nome: 1, email: 1, _id: 0 }
);
```

13.5 Denormalização vs. Normalização

Exemplo Desnormalizado (frequente em MongoDB):

```
{
  _id: ObjectId(...),
  nome_usuario: "João",
  departamento: {
    nome: "TI",
```

```

        localizacao: "Braga",
        orcamento: 100000
    },
    competencias: ["Python", "JavaScript"]
}

```

Exemplo Normalizado (similar a SQL):

```

// Colecção usuarios
{ _id: ObjectId(...), nome: "João", id_departamento: 1 }

// Colecção departamentos
{ _id: 1, nome: "TI", localizacao: "Braga", orcamento: 100000 }

// Colecção competencias_usuario
{ usuario_id: ObjectId(...), competencia: "Python" }

```

14. Comparação MongoDB Shell vs. SQL

Operação	SQL	MongoDB Shell
Criar Tabela	CREATE TABLE	db.createCollection()
Inserir	INSERT INTO	insertOne(), insertMany()
Consultar	SELECT	find()
Filtrar	WHERE	find(filtro)
Atualizar	UPDATE	updateOne(), updateMany()
Remover	DELETE	deleteOne(), deleteMany()
Junção	JOIN	\$lookup
Agregação	GROUP BY	\$group
Índice	CREATE INDEX	createIndex()
Transação	BEGIN TRANSACTION	startTransaction()

15. Exemplos Integrados Completos

Exemplo 1: Pipeline de Agregação Avançado

```

db.vendas.aggregate([
    // Filtrar vendas do ano 2025
    {
        $match: {
            data_venda: {
                $gte: ISODate("2025-01-01"),

```

```

        $lt: ISODate("2026-01-01")
    }
}
},
// Desconstrutando array de produtos
{
    $unwind: "$produtos"
},
// Juntar com coleção de produtos
{
    $lookup: {
        from: "produtos",
        localField: "produtos.id_produto",
        foreignField: "_id",
        as: "detalhes_produto"
    }
},
// Agrupar por categoria
{
    $group: {
        _id: "$detalhes_produto.categoria",
        total_vendas: { $sum: "$produtos.quantidade" },
        receita_total: { $sum: "$produtos.preco_unitario" },
        numero_transacoes: { $sum: 1 }
    }
},
// Ordenar por receita
{
    $sort: { receita_total: -1 }
},
// Limitar top 5
{
    $limit: 5
},
// Reformatar resultado
{
    $project: {
        categoria: "$_id",
        total_vendas: 1,
        receita_total: { $round: ["$receita_total", 2] },
        numero_transacoes: 1,
        _id: 0
    }
}
]);

```

Exemplo 2: CRUD Completo com Validação

```

// 1. Criar coleção com validação
db.createCollection("empleados", {
    validator: {

```

```

$jsonSchema: {
    bsonType: "object",
    required: ["nome", "email", "departamento"],
    properties: {
        nome: { bsonType: "string" },
        email: {
            bsonType: "string",
            pattern: "^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.\.[a-zA-Z]{2,}$"
        },
        idade: { bsonType: "int", minimum: 18 },
        salario: { bsonType: "decimal", minimum: 0 },
        departamento: { bsonType: "string" }
    }
}
});

// 2. Criar índices
db.empleados.createIndex({ email: 1 }, { unique: true });
db.empleados.createIndex({ departamento: 1 });

// 3. Inserir
db.empleados.insertMany([
    { nome: "João", email: "joao@example.com", departamento: "TI", salario: NumberDecimal("3500") },
    { nome: "Maria", email: "maria@example.com", departamento: "RH", salario: NumberDecimal("3000") }
]);

// 4. Consultar
db.empleados.find({ departamento: "TI" });

// 5. Atualizar
db.empleados.updateOne(
    { email: "joao@example.com" },
    { $set: { salario: NumberDecimal("3800") } }
);

// 6. Remover
db.empleados.deleteOne({ email: "maria@example.com" });

```

Exemplo 3: Transação com Rollback

```

session = db.getMongo().startSession();

try {
    session.startTransaction();

    // Transferência de fundos
    db.contas.updateOne(
        { numero_conta: "001" },

```

```
{ $inc: { saldo: -500 } },
{ session: session }
);

// Simular erro
if (Math.random() < 0.5) {
  throw new Error("Erro simulado");
}

db.contas.updateOne(
  { numero_conta: "002" },
  { $inc: { saldo: 500 } },
  { session: session }
);

session.commitTransaction();
print("Transferência realizada com sucesso");
} catch (error) {
  session.abortTransaction();
  print("Transferência cancelada: " + error.message);
} finally {
  session.endSession();
}
```

16. Conclusão

O MongoDB Shell oferece uma interface JavaScript completa para gerir bases de dados documentais. Ao contrário do SQL relacional, MongoDB privilegia flexibilidade e escalabilidade horizontal. Os conceitos fundamentais incluem:

1. **Documentos JSON** em vez de linhas
2. **Collections** em vez de tabelas
3. **MQL (MongoDB Query Language)** em vez de SQL
4. **Índices** para performance
5. **Agregação** para análise complexa
6. **Transações** para garantir ACID em múltiplos documentos
7. **Validação de Schema** para integridade de dados

A compreensão profunda da notação, sintaxe e semântica do MongoDB Shell é essencial para engenheiros informáticos e analistas de dados que trabalham com bases de dados NoSQL modernas.