

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Pedro Lima Guarnieri

**Estudo de Caso Comparando o MongoDB e o
PostgreSQL na Realização de Consultas
Agregadas de Dados**

Uberlândia, Brasil

2025

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Pedro Lima Guarnieri

**Estudo de Caso Comparando o MongoDB e o
PostgreSQL na Realização de Consultas Agregadas de
Dados**

Trabalho de conclusão de curso apresentado
à Faculdade de Computação da Universidade
Federal de Uberlândia, como parte dos requi-
sitos exigidos para a obtenção título de Ba-
charel em Ciência da Computação.

Orientador: Maria Camila Nardini Barioni

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2025

Resumo

Quando se trata de comparações de técnicas de consulta analítica em bancos de dados, um dos desafios centrais é garantir equivalência semântica entre consultas escritas em paradigmas distintos e, ao mesmo tempo, avaliar seu desempenho sob modelos de dados variados. Nesse contexto, o presente trabalho abordou um estudo de caso voltado à implementação e comparação de consultas agregadas em PostgreSQL (sistema de gerenciamento de banco de dados relacional) e MongoDB (sistema de gerenciamento de banco de dados orientado a documentos). Foram propostas consultas analíticas equivalentes empregando construtores SQL como funções agregadas (`COUNT`, `SUM`, `AVG`, `MIN`, `MAX`), `GROUP BY` e `JOIN`, e os estágios no aggregation pipeline do MongoDB e validada a correspondência semântica entre as respostas geradas por ambos os sistemas. O método de trabalho incluiu a adoção de modelos de dados normalizados e desnormalizados, a instrumentação de planos de execução (por meio de `EXPLAIN ANALYZE` no PostgreSQL e do método `explain("executionStats")` nas consultas e pipelines de agregação do MongoDB) para mensuração de métrica de desempenho (tempos de execução). Após a execução dos experimentos, verificou-se que o desempenho depende fortemente do modelo de dados e do padrão de acesso: modelos normalizados favorecem consultas que exploram junções bem indexadas, enquanto modelos desnormalizados e pipelines otimizados podem reduzir a necessidade de junções em cenários de leitura intensiva. Conclui-se que o PostgreSQL provê um fundamento comparativo robusto para validação semântica e que a escolha entre SGBD relacional e orientado a documentos deve considerar o perfil de consultas, requisitos de consistência e objetivos de escalabilidade.

Palavras-chave: SQL, agregação, PostgreSQL, MongoDB, desempenho.

Lista de ilustrações

Figura 1 – Esquema relacional (3FN) para e-commerce: Categoria, Produto, Pedido e ItemPedido. Fonte: autoria própria	18
Figura 2 – Documento JSON de um <i>pedido</i> com itens embutidos (inclui <code>categoria_id</code> no item). Fonte: autoria própria	20
Figura 3 – Scheme tree simplificado para a coleção pedidos : o nó raiz (Pedido) contém o array <code>itens[]</code> de subdocumentos, cada um com <code>produto_id</code> , <code>quantidade</code> e <code>valor</code>	21
Figura 4 – Representação da Scheme tree no formato utilizado na Modelagem Não Relacional.	21
Figura 5 – Consulta agregada SQL por categoria no domínio de e-commerce. Fonte: autoria própria	22
Figura 6 – Consulta agregada MQL equivalente à SQL, usando itens embutidos com <code>categoria_id</code> . Fonte: autoria própria	24
Figura 7 – Modelo Relacional. Fonte: autoria própria	30
Figura 8 – Comando COPY. Fonte: autoria própria	30
Figura 9 – Tabela temporária do <code>order_reviews</code> . Fonte: autoria própria	31
Figura 10 – Configuração do ambiente . Fonte: autoria própria	31
Figura 11 – Função do <code>datetime</code> . Fonte: autoria própria	32
Figura 12 – Comando <code>parse_dates</code> . Fonte: autoria própria	32
Figura 13 – Montagem do documento. Fonte: autoria própria	33
Figura 14 – Modelo com nove coleções. Fonte: autoria própria	34
Figura 15 – Modelo com duas coleções. Fonte: autoria própria	35
Figura 16 – Modelo com uma coleção. Fonte: autoria própria	35
Figura 17 – Consulta Agregada no PostgreSQL. Fonte: autoria própria	36
Figura 18 – Pipeline de Agregação no MongoDB com 9 coleções. Fonte: autoria própria	37
Figura 19 – Pipeline de Agregação no MongoDB com duas coleções. Fonte: autoria própria	37
Figura 20 – Pipeline de Agregação no MongoDB com uma coleção. Fonte: autoria própria	38
Figura 21 – Consulta Agregada no PostgreSQL. Fonte: autoria própria	38
Figura 22 – Pipeline de Agregação no MongoDB com nove coleções. Fonte: autoria própria	39
Figura 23 – Pipeline de Agregação no MongoDB com duas coleções. Fonte: autoria própria	40

Figura 24 – Pipeline de Agregação no MongoDB com uma coleção. Fonte: autoria própria	41
Figura 25 – Consulta Agregada no PostgreSQL. Fonte: autoria própria	41
Figura 26 – Pipeline de Agregação no MongoDB com nove coleções. Fonte: autoria própria	42
Figura 27 – Pipeline de Agregação no MongoDB com duas coleções. Fonte: autoria própria	43
Figura 28 – Pipeline de Agregação no MongoDB com uma coleção. Fonte: autoria própria	43
Figura 29 – Consulta Agregada no PostgreSQL. Fonte: autoria própria	44
Figura 30 – Pipeline de Agregação no MongoDB com nove coleções. Fonte: autoria própria	44
Figura 31 – Pipeline de Agregação no MongoDB com duas coleções. Fonte: autoria própria	45
Figura 32 – Pipeline de Agregação no MongoDB com uma coleção. Fonte: autoria própria	46
Figura 33 – Consulta Agregada no PostgreSQL. Fonte: autoria própria	46
Figura 34 – Pipeline de Agregação no MongoDB com nove coleções. Fonte: autoria própria	47
Figura 35 – Pipeline de Agregação no MongoDB com duas coleções. Fonte: autoria própria	48
Figura 36 – Pipeline de Agregação no MongoDB com uma coleção. Fonte: autoria própria	48
Figura 37 – Consulta Agregada no PostgreSQL. Fonte: autoria própria	49
Figura 38 – Pipeline de Agregação no MongoDB com nove coleções. Fonte: autoria própria	50
Figura 39 – Pipeline de Agregação no MongoDB com duas coleções. Fonte: autoria própria	51
Figura 40 – Pipeline de Agregação no MongoDB com uma coleção. Fonte: autoria própria	51
Figura 41 – Consulta Agregada no PostgreSQL. Fonte: autoria própria	52
Figura 42 – Pipeline de Agregação no MongoDB com nove coleções. Fonte: autoria própria	53
Figura 43 – Pipeline de Agregação no MongoDB com duas coleções. Fonte: autoria própria	54
Figura 44 – Pipeline de Agregação no MongoDB com uma coleção. Fonte: autoria própria	54
Figura 45 – Consulta Agregada no PostgreSQL. Fonte: autoria própria	55

Figura 46 – Pipeline de Agregação no MongoDB com nove coleções. Fonte: autoria própria	56
Figura 47 – Pipeline de Agregação no MongoDB com duas coleções. Fonte: autoria própria	57
Figura 48 – Pipeline de Agregação no MongoDB com uma coleção. Fonte: autoria própria	58
Figura 49 – Consulta Agregada no PostgreSQL. Fonte: autoria própria	59
Figura 50 – Pipeline de Agregação no MongoDB com nove coleções. Fonte: autoria própria	60
Figura 51 – Pipeline de Agregação no MongoDB com duas coleções. Fonte: autoria própria	61
Figura 52 – Pipeline de Agregação no MongoDB com uma coleção. Fonte: autoria própria	62
Figura 53 – Consulta Agregada no PostgreSQL. Fonte: autoria própria	63
Figura 54 – Pipeline de Agregação no MongoDB com nove coleções. Fonte: autoria própria	64
Figura 55 – Pipeline de Agregação no MongoDB com duas coleções. Fonte: autoria própria	65
Figura 56 – Pipeline de Agregação no MongoDB com uma coleção. Fonte: autoria própria	66
Figura 57 – Total de Vendas por Estado do Cliente. Fonte: autoria própria	68
Figura 58 – Produtos Mais Vendidos (Top 10). Fonte: autoria própria	68
Figura 59 – Top 5 Categorias mais Vendidas por Número de Pedidos. Fonte: autoria própria	69
Figura 60 – Top 5 Categorias mais Vendidas por Número de Pedidos. Fonte: autoria própria	69
Figura 61 – Pedidos Pagos em Parcela Única. Fonte: autoria própria	70
Figura 62 – Pedidos Pagos em Parcela Única. Fonte: autoria própria	70
Figura 63 – Pedidos Pagos em Parcela Única. Fonte: autoria própria	70
Figura 64 – Resultado da última consulta no PostgreSQL	71
Figura 65 – Resultado da última consulta no MongoDB	71
Figura 66 – Total de Vendas por Estado do Cliente. Fonte: autoria própria	82
Figura 67 – Produtos Mais Vendidos (Top 10). Fonte: autoria própria	82
Figura 68 – Receita Total por Categoria de Produto - Parte 1. Fonte: autoria própria	83
Figura 69 – Receita Total por Categoria de Produto - Parte 2. Fonte: autoria própria	83
Figura 70 – Receita Total por Categoria de Produto - Parte 3. Fonte: autoria própria	83

Figura 71 – Top 10 Vendedores por Número de Itens Vendidos. Fonte: autoria própria	84
Figura 72 – Top 5 Categorias mais Vendidas por Número de Pedidos. Fonte: autoria própria	84
Figura 73 – Análise de Estados com Maior Número de Pedidos em Parcelas Acima de 12 Vezes. Fonte: autoria própria	85
Figura 74 – Pedidos Pagos em Parcela Única - Parte 1. Fonte: autoria própria . . .	85
Figura 75 – Pedidos Pagos em Parcela Única - Parte 2. Fonte: autoria própria . . .	86
Figura 76 – Número de Pedidos Bem-Sucedidos e Valor Total por Método de Pagamento. Fonte: autoria própria	86
Figura 77 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - Parte 1. Fonte: autoria própria	87
Figura 78 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - Parte 2. Fonte: autoria própria	87
Figura 79 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - Parte 3. Fonte: autoria própria	88
Figura 80 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - Parte 4. Fonte: autoria própria	88
Figura 81 – Top 5 Categorias por Receita e Tempo Médio de Entrega. Fonte: autoria própria	88
Figura 82 – Total de Vendas por Estado do Cliente - parte 1. Fonte: autoria própria	90
Figura 83 – Total de Vendas por Estado do Cliente - parte 2. Fonte: autoria própria	90
Figura 84 – Total de Vendas por Estado do Cliente - parte 3. Fonte: autoria própria	90
Figura 85 – Produtos Mais Vendidos (Top 10) - parte 1. Fonte: autoria própria . .	91
Figura 86 – Produtos Mais Vendidos (Top 10) - parte 2. Fonte: autoria própria . .	91
Figura 87 – Receita Total por Categoria de Produto - Parte 1. Fonte: autoria própria	92
Figura 88 – Receita Total por Categoria de Produto - Parte 2. Fonte: autoria própria	92
Figura 89 – Receita Total por Categoria de Produto - Parte 3. Fonte: autoria própria	92
Figura 90 – Top 10 Vendedores por Número de Itens Vendidos - parte 1. Fonte: autoria própria	93
Figura 91 – Top 10 Vendedores por Número de Itens Vendidos - parte 2. Fonte: autoria própria	93
Figura 92 – Top 5 Categorias mais Vendidas por Número de Pedidos. Fonte: autoria própria	93
Figura 93 – Regiões onde Residem os Usuários com o Maior Número de Parcelas - parte 1. Fonte: autoria própria	94

Figura 94 – Regiões onde Residem os Usuários com o Maior Número de Parcelas - parte 2. Fonte: autoria própria	94
Figura 95 – Regiões onde Residem os Usuários com o Maior Número de Parcelas - parte 3. Fonte: autoria própria	94
Figura 96 – Pedidos Pagos em Parcela Única - Parte 1. Fonte: autoria própria . . .	95
Figura 97 – Pedidos Pagos em Parcela Única - Parte 2. Fonte: autoria própria . . .	95
Figura 98 – Pedidos Pagos em Parcela Única - Parte 3. Fonte: autoria própria . . .	95
Figura 99 – Número de Pedidos Bem-Sucedidos e Valor Total por Método de Pa- gamento. Fonte: autoria própria	96
Figura 100 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento. Fonte: autoria própria	96
Figura 101 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - parte 1. Fonte: autoria própria	97
Figura 102 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - parte 2. Fonte: autoria própria	97
Figura 103 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - parte 3. Fonte: autoria própria	98
Figura 104 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - parte 4. Fonte: autoria própria	98
Figura 105 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - parte 5. Fonte: autoria própria	99
Figura 106 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - parte 6. Fonte: autoria própria	99
Figura 107 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - parte 7. Fonte: autoria própria	100
Figura 108 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - parte 8. Fonte: autoria própria	100
Figura 109 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - parte 9. Fonte: autoria própria	101
Figura 110 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - parte 10. Fonte: autoria própria	101
Figura 111 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - parte 11. Fonte: autoria própria	102
Figura 112 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - parte 12. Fonte: autoria própria	102
Figura 113 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - parte 13. Fonte: autoria própria	103
Figura 114 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - parte 14. Fonte: autoria própria	103

Figura 115–Receita e Quantidade de Pedidos por Estado e Método de Pagamento	
- parte 15. Fonte: autoria própria	104
Figura 116–Receita e Quantidade de Pedidos por Estado e Método de Pagamento	
- parte 16. Fonte: autoria própria	104
Figura 117–Receita e Quantidade de Pedidos por Estado e Método de Pagamento	
- parte 17. Fonte: autoria própria	105
Figura 118–Receita e Quantidade de Pedidos por Estado e Método de Pagamento	
- parte 18. Fonte: autoria própria	105
Figura 119–Receita e Quantidade de Pedidos por Estado e Método de Pagamento	
- parte 19. Fonte: autoria própria	106
Figura 120–Receita e Quantidade de Pedidos por Estado e Método de Pagamento	
- parte 20. Fonte: autoria própria	106
Figura 121 –Top 5 Categorias por Receita e Tempo Médio de Entrega. Fonte: autoria própria	107

Lista de tabelas

Tabela 1 – Equivalência dos comandos CRUD entre SQL e MQL. Fonte: autoria própria	25
Tabela 2 – Tempos médios de execução das consultas.	73

Lista de abreviaturas e siglas

ACID	Atomicidade, Consistência, Isolamento e Durabilidade (propriedades de transações em SGBDs relacionais)
API	Application Programming Interface (Interface de Programação de Aplicações)
JSON	Binary JSON (formato binário usado pelo MongoDB para armazenar documentos)
CRUD	Create, Read, Update, Delete (operações básicas sobre dados)
CSV	Comma-Separated Values (formato de arquivo para intercâmbio de dados)
ETL	Extract, Transform, Load (processo de extração, transformação e carregamento de dados)
GIN	Generalized Inverted Index (tipo de índice do PostgreSQL, frequentemente usado com jsonb)
JSON	JavaScript Object Notation (formato textual de representação de dados, amplamente usado em bancos orientados a documentos)
MQL	MongoDB Query Language (linguagem de consulta do MongoDB)
NoSQL	Not Only SQL (categoria de bancos que não seguem estritamente o modelo relacional tradicional)
OLAP	Online Analytical Processing (processamento analítico, foco em consultas agregadas e análises históricas)
OLTP	Online Transaction Processing (processamento de transações on-line, foco em operações rápidas e consistentes)
RDBMS	Relational Database Management System (sistema de gerenciamento de bancos de dados baseado no modelo relacional)
SGBD	Sistema de Gerenciamento de Banco de Dados (software para definição, manipulação e consulta de bases de dados)
SGBDR	Sistema Gerenciador de Banco de Dados Relacional (variante específica de SGBD)

SQL	Structured Query Language (linguagem padrão para consulta e manipulação de dados em SGBDs relacionais)
YCSB	Yahoo! Cloud Serving Benchmark (benchmark para avaliação de performance de sistemas de armazenamento)
DBaaS	Database-as-a-Service (banco de dados oferecido como serviço em nuvem)

Sumário

1	INTRODUÇÃO	14
1.1	Objetivos	14
1.2	Justificativa	15
1.3	Organização do Texto	15
2	CONCEITOS FUNDAMENTAIS SOBRE BANCO DE DADOS	17
2.1	Modelo de Banco Relacional	17
2.2	Modelo Orientado a Documentos	19
2.2.1	Notação em Modelagem Documental	20
2.3	Linguagem SQL	21
2.4	Linguagem MQL	23
3	TRABALHOS CORRELATOS	26
3.1	Estudos Comparativos entre Bancos Relacionais e Bancos Não Relacionais	26
3.2	Modelagem orientada a documentos no PostgreSQL	26
3.3	Modelagem orientada a documentos no MongoDB	27
3.4	Pipelines de agregação e consultas analíticas	27
3.5	Trabalhos com dados de e-commerce ou conjuntos de dados semelhantes	27
3.6	Síntese de pesquisa	28
4	MÉTODO DE TRABALHO	29
4.1	Obtenção da Base de Dados	29
4.2	Modelagem Relacional e ETL no PostgreSQL	29
4.3	Modelagem Não Relacional e ETL no MongoDB	31
4.3.1	Modelagem com Nove Coleções	33
4.3.2	Modelagem com Duas Coleções	34
4.3.3	Modelagem com Uma Coleção	35
4.4	Desenvolvimento das Consultas Agregadas	36
4.4.1	Total de Vendas por Estado do Cliente	36
4.4.2	Produtos Mais Vendidos (Top 10)	38
4.4.3	Receita Total por Categoria de Produto	41
4.4.4	Top 10 Vendedores por Número de Itens Vendidos	43
4.4.5	Top 5 Categorias mais Vendidas por Número de Pedidos	46

4.4.6	Análise de Estados com Maior Número de Pedidos em Parcelas Acima de 12 Vezes	49
4.4.7	Pedidos Pagos em Parcela Única	51
4.4.8	Número de Pedidos Bem-Sucedidos e Valor Total por Método de Pagamento	55
4.4.9	Receita e Quantidade de Pedidos por Estado e Método de Pagamento . . .	58
4.4.10	Top 5 Categorias por Receita Total, Tempo Médio de Entrega, Número de Itens e Número de Pedidos	62
5	RESULTADOS E DISCUSSÃO	67
5.1	Resultado das Consultas Seleccionadas Para Comparação	67
5.2	Comparação dos Resultados	70
5.3	Análise Interpretativa	71
5.4	Validação e Comparação de Desempenho entre PostgreSQL e MongoDB	72
5.4.1	Ambiente de Teste	72
5.4.2	Metodologia de Medição	72
5.4.3	Tabelas de Comparação	72
6	CONCLUSÕES E RECOMENDAÇÕES PARA TRABALHOS FUTUROS	74
	REFERÊNCIAS	76
	Apêndices	80
A	RESULTADOS COMPLETOS DAS CONSULTAS	81
A.1	Resultados das Consultas no PostgreSQL	81
A.2	Resultados das Consultas no MongoDB	88

1 Introdução

Sabe-se que, nos últimos anos, houve um aumento do uso de tecnologias apropriadas para lidar com grandes quantidades de informação, especialmente em ambientes que demandam escalabilidade horizontal e processamento eficiente de dados não estruturados (KUMAR et al., 2023). Nesse contexto, os sistemas gerenciadores de bancos de dados NOSQL emergiram como alternativa aos sistemas relacionais tradicionais, por oferecerem maior flexibilidade na manipulação de diversos modelos de dados como documentos, grafos e colunas e por suportarem arquiteturas escaláveis que atendem exigências modernas de desempenho e disponibilidade (LUJÁN-MORA, 2024).

Dentre as bases de dados NOSQL, o MongoDB se destaca por sua organização orientada a documentos e pelas facilidades que fornece para operações analíticas e de suporte à decisão. Seu **framework** de agregação possibilita operações complexas como resumo, filtragem, agrupamento e ordenação de dados o que o torna adequado para análises em cenários com grande volume informacional, como sistemas de recomendação, mídias sociais, e-commerce e aplicações em Internet das Coisas (MONGODB, 2024); (MISHRA; LODHI; MEHTA, 2018). Estudos comparativos indicam vantagens do MongoDB em operações de agregação e em cargas analíticas, especialmente quando é necessária escalabilidade horizontal e processamento em tempo real (GYÖRÖDI et al., 2022); (KUMAR et al., 2023); (BAJPAYEE et al., 2021).

Com o crescimento de tecnologias que geram enorme quantidade de dados por exemplo, Internet das Coisas (IoT), redes sociais e aplicações em nuvem a necessidade de ferramentas que processem e agreguem estes dados de forma eficiente torna-se crítica. Projeções recentes apontam para volumes globais de informações em zettabytes, o que evidencia a importância de arquiteturas e mecanismos de agregação capazes de suportar análises de larga escala e tomadas de decisão baseadas em dados (DOMO, 2022); (HERIPRACOY, 2016).

1.1 Objetivos

Ao se abordar o tópico consultas agregadas em sistemas gerenciadores de bancos de dados não relacionais, é possível destacar que ainda não está claramente estabelecido qual é a eficácia e a precisão desses mecanismos quando empregados para suporte à decisão em cenários reais, especialmente quando considerados conjuntos de dados volumosos e heterogêneos. Nesse mesmo sentido, o presente trabalho tem o objetivo de contribuir com essa questão comparando de forma semântica consultas analíticas entre modelos de dados normalizados e desnormalizados no sistema gerenciador de banco de dados não relacional

MongoDB e no sistema gerenciador de banco de dados relacional PostgreSQL e avaliar métricas de desempenho, a fim de demonstrar equivalência nos resultados retornados e sua eficiência no tratamento de suporte à decisão por meio de consultas analíticas.

1.2 Justificativa

A pesquisa proposta aqui é importante para apoiar futuros estudos que possam utilizar consultas agregadas em sistema gerenciadores de banco de dados não relacional NoSQL, como o MongoDB, independentemente da área científica ou aplicação prática a ser explorada. Sabe-se que, nem todos os conjuntos de dados armazenados em bases orientadas a documentos apresentam esquemas rígidos ou campos estruturados de forma uniforme, informações relevantes para análises agregadas podem estar dispersas, incompletas ou registradas em formatos heterogêneos. Tendo em vista tal situação, verifica-se necessário o desenvolvimento de um método de avaliação da eficiência e da confiabilidade das consultas agregadas, de modo a quantificar sua precisão e robustez ao produzir indicadores e resumos a partir de coleções com grande variabilidade estrutural. Nesse sentido, estudos que pretendem avaliar a qualidade das análises e a correlação entre resultados agregados e a realidade observada serão os principais beneficiados.

No contexto deste estudo, deve-se considerar a possível indisponibilidade ou inconsistência de informações nos campos que sustentam operações analíticas — por exemplo, ausência de campos-chave, nomes de atributos divergentes entre coleções e tabelas, valores nulos ou formatos distintos para a mesma informação. Contudo, o método adotado foca na comparação semântica e na avaliação de desempenho entre consultas SQL em PostgreSQL e pipelines de agregação no MongoDB, validando a equivalência dos resultados (contagens, somas e médias) e medindo latência. Já investigações mais aprofundadas sobre a robustez das agregações frente a inconsistências estruturais (por exemplo, simulações de remoção de campos, uso de valores pré-agregados ou índices parciais) são apontadas como trabalhos futuros.

1.3 Organização do Texto

O texto deste trabalho é organizado da seguinte forma. O Capítulo 2, Conceitos Fundamentais sobre Banco de Dados, apresenta, de forma geral, como são realizadas as pesquisas e análises envolvendo bancos de dados orientados a documentos e banco de dados relacionais, com destaque para os conceitos necessários para o entendimento da pesquisa realizada no trabalho de conclusão de curso que é descrita aqui. O Capítulo 3, Trabalhos Correlatos, aborda pesquisas de outros autores que também envolveram alguns dos temas relatados aqui. Em seguida, no Capítulo 4, descreve-se o Método de Trabalho desta pesquisa, adotado com o intuito de elaborar o **framework** previsto e possibilitar sua

aplicação em uma base de dados real, representativa de um problema do mundo real. O Capítulo 5 Resultados e Discussões, apresenta, primeiramente, os resultados obtidos nas duas modelagens empregadas. Em seguida, apresenta toda a discussão de tais resultados, juntamente com a análise crítica sobre implicações para suporte à decisão e possíveis limitações identificadas. Além disso, conclui-se o trabalho com os novos conhecimentos adquiridos e, além disso, recomendações para a continuidade desta pesquisa. Finalmente, a monografia conta com um Apêndice que é apresentado no capítulo final, no qual são detalhados todos os resultados das consultas utilizadas.

2 Conceitos Fundamentais sobre Banco de Dados

O presente trabalho aborda um estudo de caso sobre consultas agregadas utilizando MongoDB e PostgreSQL. Tal estudo permite comparar semanticamente as consultas de agregação nos dois sistemas gerenciadores de banco de dados e avaliar o desempenho entre modelos de dados normalizados e desnormalizados. Pode-se, por exemplo, examinar como os operadores e **pipelines** de agregação do MongoDB se correlacionam com consultas SQL com **GROUP BY**, **JOIN** e funções agregadas no PostgreSQL, e de que forma essas diferenças impactam expressividade, equivalência semântica e os resultados práticos em tempo de execução. Trata-se, nesse caso, de uma investigação sobre modelagem de dados orientada a documentos em contraste com a modelagem relacional de dados. Essas abordagens de modelagem apresentam características distintas que influenciam desde a forma de representar até estratégias de indexação e de manutenção da consistência. Os conceitos citados aqui serão abordados ao longo do trabalho e, dessa forma, torna-se necessário elucidá-los.

Diferentemente dos modelos relacionais estritamente normalizados, cujas regras de integridade e esquemas rígidos são bem definidos, a modelagem orientada a documentos permite esquemas flexíveis e aninhamento de dados, o que favorece a desnormalização e consultas mais diretas em alguns cenários. Além disso, com a variedade de padrões de acesso e requisitos de consistência, tais modelos demandam escolhas que impactam índices, particionamento e mecanismos de escrita e leitura. Outro aspecto é que não existe uma única forma padrão de traduzir consultas agregadas entre diferentes sistemas de banco de dados, o que faz com que a equivalência entre **pipelines** do MongoDB e consultas SQL nem sempre seja simples, exigindo transformações e adaptações no projeto dos dados e nas consultas para garantir resultados e desempenho comparáveis.

Em geral, o processo completo de avaliação comparativa é composto pelas seguintes etapas: Modelo Relacional, Modelo Orientado a Documentos, Linguagem SQL e Linguagem MQL. Tendo em vista essas etapas e o método de trabalho adotado na pesquisa, elaborou-se uma revisão bibliográfica que aborda cada uma das fases, relacionando-as às técnicas e boas práticas aplicadas no experimento.

2.1 Modelo de Banco Relacional

O modelo relacional de dados, proposto por E. F. Codd em 1970, representa a informação como um conjunto de relações formalmente estruturadas em tabelas bidimen-

sionais compostas por linhas (tuplas) e colunas (atributos). Cada relação deve possuir uma chave primária como pode ser visto na Figura 1. Esta chave tem a função de identificar unicamente cada tupla, assegurando a integridade da entidade. Adicionalmente, o modelo estabelece a restrição de unicidade da chave, de modo que não existam duplicidades em atributos definidos como identificadores. Os relacionamentos entre entidades distintas são representados por meio de chaves estrangeiras, sujeitas à restrição de integridade referencial, a qual garante que os valores atribuídos a essas chaves correspondam a tuplas existentes na relação de referência. Por fim, cada atributo de uma relação está associado a um domínio, isto é, um conjunto de valores válidos, o que constitui a restrição de domínio, fundamental para a consistência semântica dos dados. (Codd, 1970); (Elmasri & Navathe, 2016).

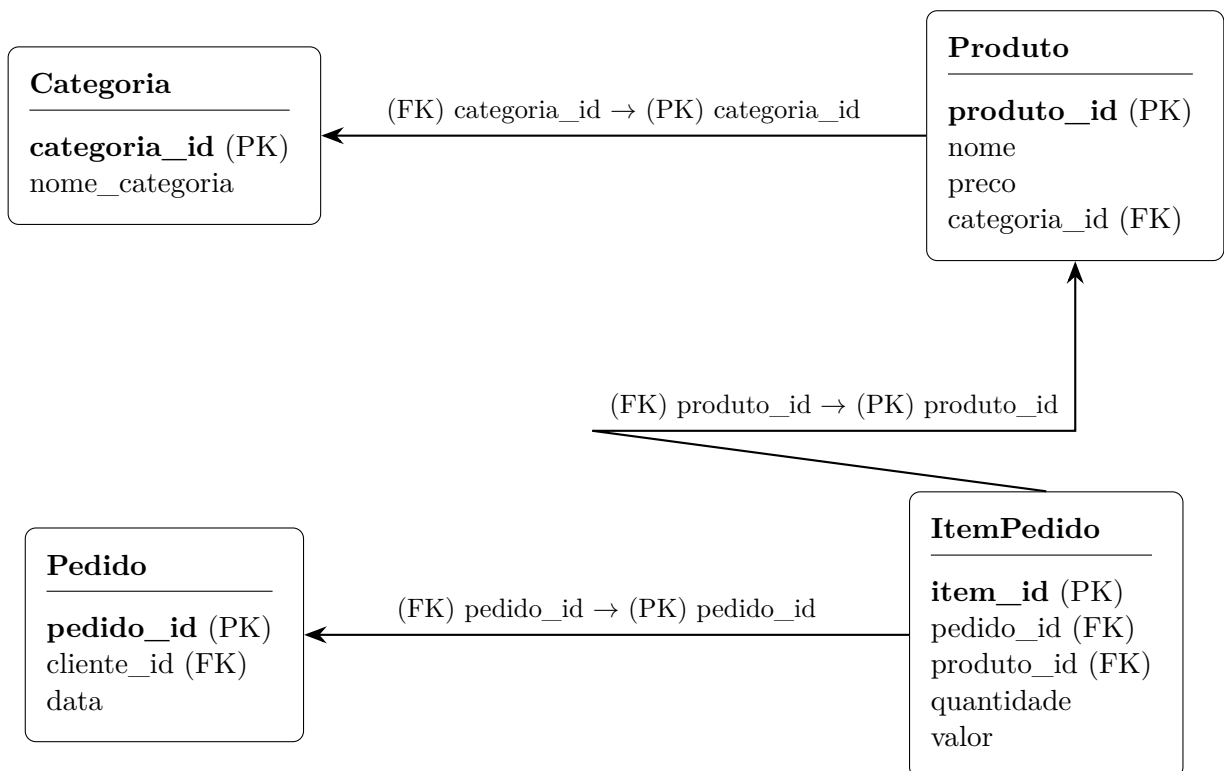


Figura 1 – Esquema relacional (3FN) para e-commerce: Categoria, Produto, Pedido e ItemPedido. Fonte: autoria própria

De acordo com Silberschatz e Cardoso, essas restrições formam a base para a confiabilidade, a consistência e a coerência lógica dos bancos de dados relacionais. (Silberschatz, 2020); (Cardoso, 2012).

Essa forma de organização pressupõe a existência de um esquema rígido, no qual cada coluna possui um tipo de dado definido. Tal rigidez garante consistência e integridade, mas implica em menor flexibilidade para lidar com dados heterogêneos (Corbellini et al., 2017). Além disso, os sistemas gerenciadores de banco de dados relacionais (SGBDRs) seguem as propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade),

assegurando confiabilidade transacional mesmo diante de falhas ou acessos concorrentes (Gray & Reuter, 1993).

A normalização é um processo fundamental no projeto de bancos de dados relacionais, cujo objetivo é organizar os dados de forma a reduzir redundâncias e evitar anomalias de inserção, atualização e exclusão. Esse processo é realizado por meio de formas normais (FN), que representam diferentes níveis de refinamento do esquema relacional:

- **Primeira Forma Normal (1FN):** exige que os atributos sejam atômicos, ou seja, cada coluna deve conter valores indivisíveis, sem listas ou repetições no mesmo campo.
- **Segunda Forma Normal (2FN):** além de atender à 1FN, requer que todos os atributos não-chave dependam da chave primária inteira, eliminando dependências parciais em chaves compostas.
- **Terceira Forma Normal (3FN):** além de cumprir a 2FN, uma relação está em 3FN se, para toda dependência funcional $X \rightarrow A$, X é superchave ou A é atributo primo. Em termos práticos, nenhum atributo não-primo pode depender transitivamente de uma chave candidata.

No contexto deste trabalho, o modelo relacional é aplicado por meio do PostgreSQL, que armazena os dados em múltiplas tabelas normalizadas ou não normalizadas. A modelagem foi realizada a partir da conversão das entidades e relacionamentos em tabelas distintas, vinculadas por chaves primárias e estrangeiras. Esse modelo favorece a execução de consultas agregadas por meio de junções e operações como `COUNT`, `SUM` e `AVG`, em combinação com a cláusula `GROUP BY`, permitindo extrair indicadores resumidos de forma consistente.

2.2 Modelo Orientado a Documentos

O modelo orientado a documentos organiza a informação em documentos semi-estruturados, tipicamente no formato JSON (JavaScript Object Notation) ou BSON (Binary JSON) armazenados em coleções, como pode ser visto na Figura 2. Cada documento é uma estrutura de pares chave-valor que pode conter subdocumentos e `arrays` aninhados, permitindo representar naturalmente estruturas hierárquicas e listas. Uma característica distintiva desse modelo é a flexibilidade de esquema: documentos dentro de uma mesma coleção podem ter atributos diferentes, o que facilita a ingestão de dados heterogêneos e iterações rápidas no modelo de domínio. Em contrapartida, essa flexibilidade demanda decisões conscientes de modelagem, de modo a evitar inconsistências e

redundância excessiva. Estudos comparativos sobre bancos de dados NoSQL destacam justamente esses dilemas entre desempenho, escalabilidade e consistência. (Pokorný, 2018); (Han et al., 2011).

```
{
  "pedido_id": 98765,
  "cliente_id": 12345,
  "data": "2024-08-15T14:23:00Z",
  "itens": [
    {
      "produto_id": 1001,
      "categoria_id": 10,
      "nome_produto": "Fone Bluetooth",
      "quantidade": 2,
      "valor": 150.00
    },
    {
      "produto_id": 1002,
      "categoria_id": 20,
      "nome_produto": "Mouse Óptico",
      "quantidade": 1,
      "valor": 80.00
    }
  ],
  "total": 380.00
}
```

Figura 2 – Documento JSON de um *pedido* com itens embutidos (inclui *categoria_id* no item). Fonte: autoria própria

2.2.1 Notação em Modelagem Documental

Diferentemente do modelo relacional, cuja representação tradicional é feita por meio de esquemas tabulares (com atributos, chaves primárias e estrangeiras), a modelagem orientada a documentos requer uma notação capaz de expressar a natureza hierárquica e flexível dos documentos JSON/BSON. Neste trabalho, adota-se a notação de **scheme trees**, conforme proposta por Mok e pode ser visto na Figura 3. (Mok, 2024)

Um **scheme tree** é uma árvore que descreve a estrutura potencial dos documentos em uma coleção MongoDB. O nó raiz representa a entidade principal, enquanto os nós filhos correspondem a atributos, subdocumentos ou **arrays**. Essa notação permite representar explicitamente elementos característicos dos bancos de dados orientados a documentos, como.

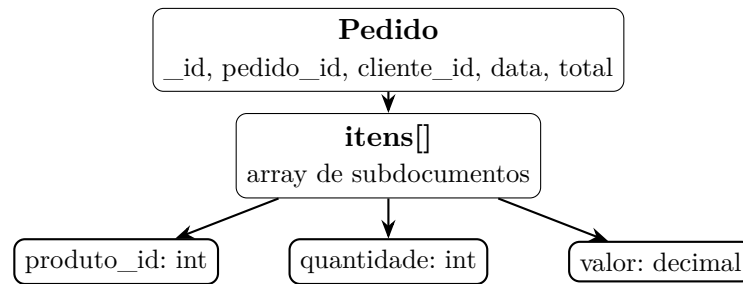


Figura 3 – Scheme tree simplificado para a coleção **pedidos**: o nó raiz (**Pedido**) contém o array **itens[]** de subdocumentos, cada um com **produto_id**, **quantidade** e **valor**.

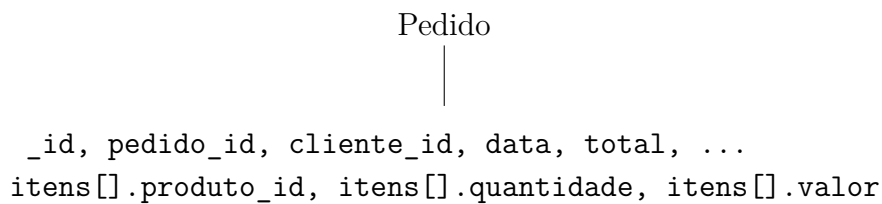


Figura 4 – Representação da Scheme tree no formato utilizado na Modelagem Não Relacional.

A principal contribuição dessa notação é permitir a comparação entre diferentes alternativas de modelagem no MongoDB. Dependendo da forma como a árvore é estruturada, é possível projetar coleções mais *normalizadas* (com maior uso de referências entre documentos) ou mais *desnormalizadas* (com embutimento de dados relacionados em um único documento). Assim, os *scheme trees* fornecem um suporte conceitual para a tomada de decisão entre *embed* e *reference*, considerando fatores como redundância, desempenho de consulta e custo de atualização.

A modelagem documental envolve equilíbrios entre armazenar dados relacionados no mesmo documento e manter documentos separados e referenciá-los por identificadores. A inserção direta reduz a necessidade de junções e pode acelerar consultas quando os mesmos grupos de dados são acessados em conjunto; já a referência preserva a normalização lógica e reduz redundância quando a cardinalidade é elevada ou as atualizações são frequentes. No experimento deste TCC foram testadas três estratégias de modelagem documental no MongoDB: nove coleções (mais próximo da normalização), duas coleções (semidesnormalizado) e uma única coleção (totalmente desnormalizado), com o objetivo de avaliar o impacto sobre desempenho, complexidade de ETL e equivalência dos resultados agregados frente ao modelo relacional.

2.3 Linguagem SQL

A **Structured Query Language** (SQL) é a linguagem padrão para consulta, definição e manipulação de dados em sistemas relacionais, com fundamentação teórica baseada

no modelo relacional de E. F. Codd. SQL engloba conjuntos de comandos utilizados para definir esquemas (DDL — CREATE, ALTER, DROP), manipular dados (DML — SELECT, INSERT, UPDATE, DELETE), controlar transações (TCL — BEGIN, COMMIT, ROLLBACK) e gerenciar privilégios (DCL — GRANT, REVOKE), como pode ser visto na Figura 5. Essas categorias e seu papel no controle de consistência e integridade transacional são discutidas em manuais clássicos de sistemas de banco de dados. (Codd et al., 1970); (Silberschatz, 2020); (Cardoso, 2012)

Para análise de dados, os construtores essenciais em SQL são as funções agregadas (COUNT, SUM, AVG, MIN, MAX), a cláusula GROUP BY (com possível filtro via HAVING) e as junções (JOIN) que permitem combinar informações de múltiplas relações. As junções podem assumir várias semânticas (INNER JOIN, LEFT/RIGHT OUTER JOIN, FULL OUTER JOIN, CROSS JOIN) e são executadas por algoritmos diferentes, cuja escolha pelo usuário afeta significativamente o desempenho das consultas analíticas. (Silberschatz, 2020); (Cardoso, 2012)

```
SELECT
  c.categoria_id,
  COUNT(*) AS qtd_vendas,
  SUM(ip.valor) AS total_vendas,
  AVG(ip.valor) AS media_venda
FROM item_pedido AS ip
JOIN pedido AS p ON p.pedido_id = ip.pedido_id
JOIN produto AS pr ON pr.produto_id = ip.produto_id
JOIN categoria AS c ON c.categoria_id = pr.categoria_id
WHERE p.data BETWEEN '2024-01-01' AND '2024-12-31'
GROUP BY c.categoria_id
HAVING COUNT(*) > 10
ORDER BY total_vendas DESC;
```

Figura 5 – Consulta agregada SQL por categoria no domínio de e-commerce. Fonte: autoria própria

Construções de organização e legibilidade de consultas, como *subqueries* correlacionadas e WITH (Common Table Expressions — CTEs), são amplamente utilizadas para decompor consultas analíticas complexas; a materialização de CTEs e o impacto dessa materialização no desempenho dependem da implementação do SGBD e de sua versão. Do ponto de vista prático, o tratamento de valores nulos (NULL), a coerência semântica entre projeções e agrupamentos e a correta especificação de predicados de seleção são aspectos que influenciam diretamente a equivalência dos resultados quando se comparam agregações entre modelos relacional e documental (Cardoso, 2012).

A instrumentação por meio de planos de execução é elemento metodológico fundamental para avaliação de desempenho: comandos como EXPLAIN ANALYZE permitem obter tempos por nó de execução, possibilitando a comparação entre estimativas e valores observados. (POSTGRES GLOBAL DEVELOPMENT GROUP, 2025)

Metodologicamente, no presente trabalho SQL foi utilizado como referência funcional para as agregações: as consultas analíticas foram implementadas em PostgreSQL, validadas semanticamente contra as implementações em MongoDB e instrumentadas com `EXPLAIN ANALYZE` para coletar tempos.

Essa abordagem assegura que comparações entre o modelo relacional (PostgreSQL) e as estratégias documentais (MongoDB) considerem não apenas equivalência semântica dos resultados agregados, mas também escolhas físicas de execução que impactam latência, *throughput* e custo de ETL. (Silberschatz, 2020); (Cardoso, 2012); (POSTGRESQL GLOBAL DEVELOPMENT GROUP, 2025)

2.4 Linguagem MQL

A **MongoDB Query Language (MQL)** é a linguagem de consulta nativa do MongoDB, baseada em documentos JSON/BSON, que permite expressar consultas, projeções, atualizações e transformações de maneira coerente com o modelo baseado em documentos. Diferentemente de SQL, MQL utiliza uma sintaxe orientada a documentos e operadores específicos que refletem o armazenamento interno do MongoDB. Para operações analíticas complexas, o MongoDB oferece o **Aggregation Framework**, que processa documentos por meio de *pipelines* compostos por estágios encadeados (`$match`, `$unwind`, `$group`, `$project`, `$sort`, `$lookup`), como pode ser visto na Figura 6. Cada estágio recebe um fluxo de documentos, aplica uma transformação e repassa o resultado ao próximo estágio, permitindo que, em muitos casos, se construam agregações equivalentes às operações `GROUP BY` e `JOIN` do SQL, porém com sintaxe e características próprias do modelo baseado em documentos. A documentação oficial e guias do *Aggregation Framework* detalham estágios, melhores práticas e recomendações de otimização (aplicar `$match` cedo, projetar índices adequados, minimizar expansões desnecessárias). (MongoDB Aggregation Framework — documentação oficial); (Silva et al., 2021).

```
db.pedidos.aggregate([
  {
    $match: {
      data: {
        $gte: ISODate("2024-01-01T00:00:00Z"),
        $lte: ISODate("2024-12-31T23:59:59Z")
      }
    }
  },
  { $unwind: "$itens" },
  {
    $group: {
      _id: "$itens.categoria_id",
      qtd_vendas: { $sum: 1 },
      total_vendas: { $sum: "$itens.valor" },
      media_venda: { $avg: "$itens.valor" }
    }
  },
  { $match: { qtd_vendas: { $gt: 10 } } },
  { $sort: { total_vendas: -1 } },
  {
    $project: {
      categoria_id: "$_id",
      qtd_vendas: 1,
      total_vendas: 1,
      media_venda: 1,
      _id: 0
    }
  }
])
```

Figura 6 – Consulta agregada MQL equivalente à SQL, usando itens embutidos com categoria_id. Fonte: autoria própria

A Tabela 1 apresenta a equivalência entre os comandos CRUD em SQL e suas correspondentes operações em MQL, ilustrando como operações comuns de consulta, inserção, atualização e exclusão são expressas em cada modelo.

SQL	MQL
SELECT	find
SELECT nome, preco FROM produtos WHERE categoria = 'eletrônicos';	db.produtos.find({ categoria: "eletrônicos" }, { nome: 1, preco: 1, _id: 0 })
INSERT	insertOne / insertMany
INSERT INTO produtos (nome, preco, categoria) VALUES ('Fone Bluetooth', 150, 'eletrônicos');	db.produtos.insertOne({ nome: "Fone Bluetooth", preco: 150, categoria: "eletrônicos" })
UPDATE	updateOne / updateMany
UPDATE produtos SET preco = 140 WHERE nome = 'Fone Bluetooth';	db.produtos.updateOne({ nome: "Fone Bluetooth" }, { \$set: { preco: 140 } })
DELETE	deleteOne / deleteMany
DELETE FROM produtos WHERE nome = 'Fone Bluetooth';	db.produtos.deleteOne({ nome: "Fone Bluetooth" })

Tabela 1 – Equivalência dos comandos CRUD entre SQL e MQL. Fonte: autoria própria

Essa apresentação permite visualizar rapidamente como as operações tradicionais de bancos relacionais (SQL) são traduzidas para o modelo baseado em documentos (MQL), evidenciando diferenças de sintaxe, abordagem e flexibilidade. Operações de leitura, escrita, atualização e remoção seguem a mesma lógica semântica, mas adaptadas às características do MongoDB, como ausência de esquema rígido e uso de documentos JSON/BSON.

No escopo deste TCC, cada consulta SQL implementada no PostgreSQL teve sua reprodução em MQL, incluindo a construção de *pipelines* de agregação, para possibilitar comparação semântica dos resultados e avaliação de desempenho. A construção de *pipelines* foi orientada pelas decisões de modelagem e por regras práticas de otimização (ordem de estágios, uso de índices e operações de redução de cardinalidade).

3 Trabalhos Correlatos

Neste capítulo são apresentados estudos que investigam o desempenho e as características de bancos de dados relacionais e bancos de dados não relacionais, com destaque para a análise de consultas agregadas e diferentes estratégias de modelagem de dados. Esta seção apresenta um panorama de pesquisas relevantes que embasam o presente trabalho, organizadas por temas.

3.1 Estudos Comparativos entre Bancos Relacionais e Bancos Não Relacionais

[Kumar et al., \(2023\)](#) realizaram um estudo comparativo entre bancos relacionais, e bancos não relacionais como serviço em nuvem (DBaaS), avaliando operações CRUD e consultas agregadas sob diferentes cargas. Os resultados evidenciaram vantagens do MongoDB em consultas complexas devido à sua arquitetura orientada a documentos, reforçando seu potencial para cenários com grandes volumes de dados. De forma semelhante, [Győrödi et al. \(2022\)](#) compararam MongoDB e MySQL na gestão de aplicações de **big data**, identificando maior eficiência do MongoDB em operações que lidam com dados não estruturados, especialmente quando envolvem pipelines de agregação.

3.2 Modelagem orientada a documentos no PostgreSQL

[Michels et al., \(2018\)](#) discutem as extensões do padrão SQL:2016 para suporte nativo a JSON, destacando como o PostgreSQL implementa tipos `json` e `jsonb`, operadores de consulta e indexação GIN para manipulação eficiente de documentos. [Chasseur et al., \(2013\)](#) exploram a integração de **document stores** em sistemas relacionais, apresentando uma camada (Argo) que mapeia JSON para um SGBDR e mostrando que é possível expor a mesma superfície de programação dos bancos documentais preservando funcionalidades relacionais (JOINS, ACID). Esses trabalhos evidenciam que o PostgreSQL, embora relacional, oferece recursos robustos para modelagem orientada a documentos, permitindo abordagens híbridas que mantêm consistência transacional e suportam consultas complexas sobre dados aninhados [MICHELS et al., 2018; CHASSEUR; LI; PATEL, 2013](#)).

3.3 Modelagem orientada a documentos no MongoDB

[Mishra et al., \(2018\)](#) analisaram bancos de dados não relacionais orientados a documentos como MongoDB, ArangoDB, Elasticsearch e OrientDB por meio do benchmark [YCSB](#), avaliando throughput e tempo de execução em **workloads** de leitura, atualização e mistas. Os autores destacaram que o MongoDB oferece flexibilidade de modelagem e desempenho robusto para agregações complexas. [Heripracoy \(2016\)](#), por sua vez, demonstrou o uso de pipelines de agregação no MongoDB em um sistema de suporte à decisão, evidenciando sua eficácia para sumarização e análise de grandes volumes de dados em tempo real.

3.4 Pipelines de agregação e consultas analíticas

[Mohan et al., \(2024\)](#) avaliaram bancos NoSQL em **workloads** OLAP, comparando MongoDB, Redis, Kudu e ArangoDB. O estudo mostrou que o MongoDB apresenta desempenho competitivo em operações de agregação, especialmente para análises de **big data**, graças à sua arquitetura orientada a documentos e ao suporte nativo a operações analíticas complexas.

3.5 Trabalhos com dados de e-commerce ou conjuntos de dados semelhantes

Diversos estudos acadêmicos têm adotado bases públicas de **e-commerce** para investigar desempenho de consultas, modelagem e análises agregadas. Um exemplo diretamente relacionado é o uso do [Brazilian E-commerce Public Dataset \(Olist\)](#), amplamente explorado em pesquisas que avaliam qualidade de dados, padrões de compra e desempenho de consultas analíticas; a própria descrição do conjunto registra aproximadamente 100 mil pedidos entre 2016 e 2018, com entidades para clientes, vendedores, pedidos, itens, pagamentos, avaliações e geolocalização (Olist/Kaggle). Trabalhos recentes analisam esse conjunto de dados em contextos de mineração de dados e qualidade da informação, indicando sua adequação para estudos comparativos entre modelos normalizados e desnormalizados ([SCITEPRESS, 2022](#)).

Outros conjuntos de dados frequentemente empregados são o [Online Retail \(UCI\)](#), o [Instacart \(2017\)](#) e o [Retail Rocket](#), todos adequados para comparar esquemas normalizados e desnormalizados em cenários com relações N:M e alto volume. [Mishra et al. \(2018\)](#) avaliaram bancos orientados a documentos via YCSB, destacando a flexibilidade e o desempenho do MongoDB em agregações; [Heripracoy \(2016\)](#) demonstrou o uso de pipelines de agregação no MongoDB em sistemas de suporte à decisão, evidenciando sua

eficácia em análises em tempo real.

3.6 Síntese de pesquisa

Embora a literatura demonstre a capacidade do MongoDB em consultas agregadas e seu bom desempenho em comparação com SGBD relacionais, a maioria dos estudos concentra-se em análises gerais ou em cenários sintéticos. Trabalhos como os de [Kumar et al., \(2023\)](#) e [Györödi et al. \(2022\)](#) destacam o potencial do MongoDB frente a cargas de dados intensas, enquanto [Mishra et al., \(2018\)](#) e [Heripracoy \(2016\)](#) exploram casos práticos com agregações complexas. No entanto, poucos estudos avaliam de forma consistente o impacto de diferentes estratégias de modelagem documental (normalizada, semi-desnormalizada e totalmente desnormalizada) sobre o desempenho de pipelines de agregação, utilizando um mesmo conjunto de dados real.

Nesse contexto, o presente trabalho diferencia-se ao investigar três variações de modelagem orientada a documentos no MongoDB, comparando-as entre si e com um modelo relacional no PostgreSQL, a partir de um conjunto de dados de e-commerce de grande porte. Tal abordagem busca fornecer evidências empíricas sobre os impactos de desempenho, custo de armazenamento e complexidade de ETL em diferentes cenários de agregação.

4 Método de Trabalho

O método a ser detalhado a seguir, relata como foi o processo de criação das consultas agregadas utilizando o Sistema Gerenciador de Banco de Dados Relacional (PostgreSQL) e o Sistema Gerenciador de Banco de Dados Não Relacional (MongoDB), que vai desde a escolha da base de dados, passa pelo tratamento, conversão de tipos e verificação de integridade e termina com a modelagem e povoamento dos bancos de dados. Por fim, o desenvolvimento das consultas agregadas e validação dos resultados.

4.1 Obtenção da Base de Dados

O experimento iniciou-se com a coleta do Conjunto de Dados Públicos de E-commerce Brasileiro disponibilizado pela Olist na plataforma [Kaggle](#), que contém aproximadamente 100 mil pedidos realizados entre 2016 e 2018 em diversos **marketplaces** brasileiros. Trata-se de um conjunto amplamente utilizado na literatura e na comunidade, já disponibilizado tratamentos prévios (integração entre entidades e dicionário de dados publicamente documentado), o que favorece reprodutibilidade. Todos os arquivos foram baixados em formato CSV, totalizando nove planilhas que contêm informações detalhadas sobre clientes, vendedores, produtos, categorias, pedidos, itens de pedido, pagamentos, avaliações e geolocalização. Ainda assim, para garantir consistência entre os SGBDs, aplicou-se um processo de ingestão de dados mínimo: verificação de integridade entre chaves lógicas, padronização leve de **strings** e **parsing** de campos temporais.

4.2 Modelagem Relacional e ETL no PostgreSQL

A modelagem relacional foi definida a partir de nove tabelas (Figura 66), incluindo **customers** (contém informações sobre os clientes), **sellers** (contém informações sobre os vendedores), **products** (contém informações sobre os produtos), **product_category_name_translation** (contém a tradução das categorias de produto do inglês para o português), **orders** (contém informações sobre os pedidos), **order_items** (contém informações sobre os itens de cada pedido), **order_payments** (contém informações sobre os pagamentos dos pedidos), **order_reviews** (contém análise sobre os pedidos) e **geolocation** (contém informações sobre geolocalização). Cada entidade recebeu seus respectivos atributos e relacionamentos para garantir a integridade. A opção pelo PostgreSQL como base relacional decorre da sua aderência às propriedades ACID, da maturidade do otimizador e da instrumentação nativa via **EXPLAIN ANALYZE**, além do suporte a **JSON/JSONB**, que favorecem comparações híbridas com dados aninhados. Adicionalmente, a escolha foi motivada pela

experiência prévia do autor com o SGBD, o que reduziu o custo de configuração, depuração e medição, garantindo maior confiabilidade aos experimentos e à reprodutibilidade.

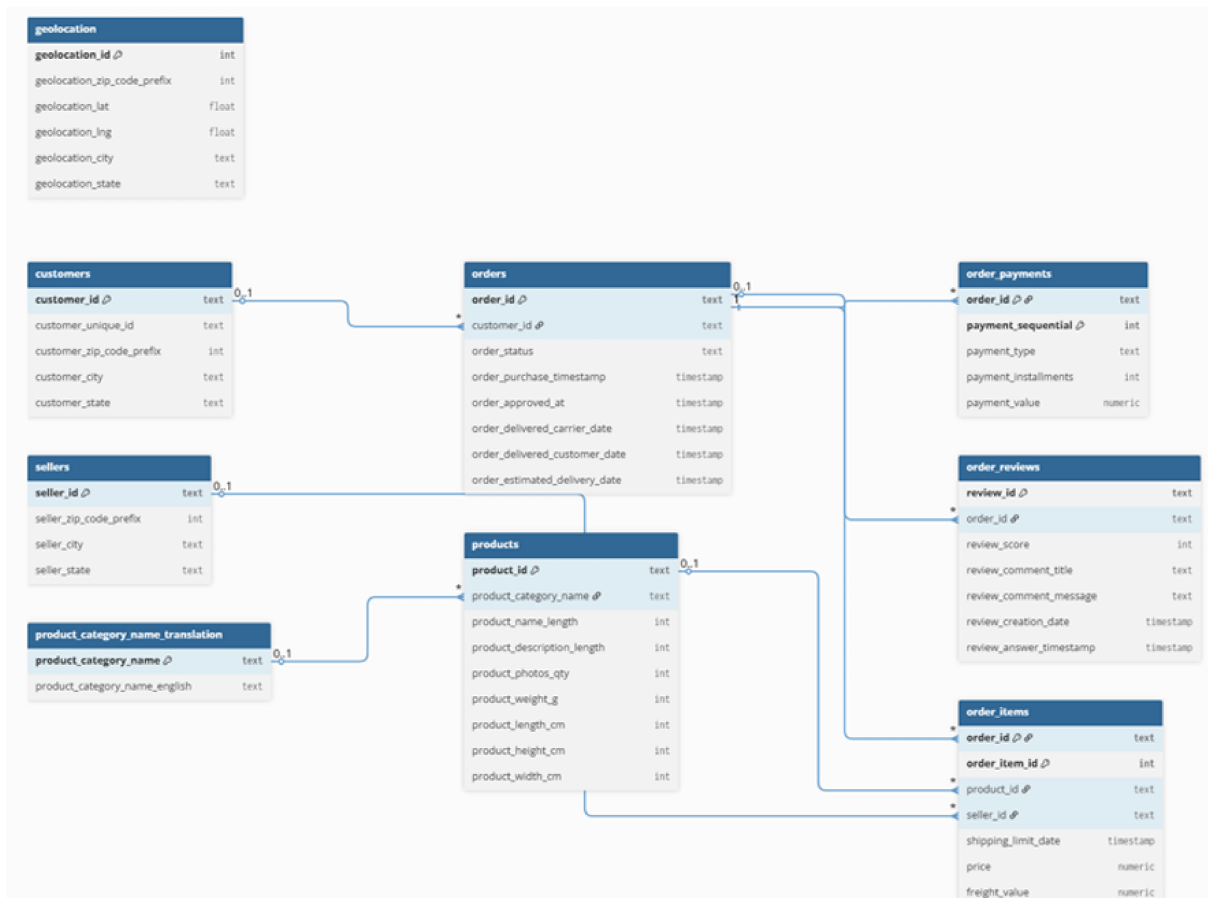


Figura 7 – Modelo Relacional. Fonte: autoria própria

Para popular o banco, foi utilizado um script SQL de ETL que utilizou o comando COPY para inserir os dados dos arquivos CSV em cada tabela (Figura 67).

```
-- Copiando os dados do dataset para CLIENTES
COPY olist.customers FROM 'C:\Users\kastr\OneDrive\Documents\TCC\olist_customers_dataset.csv' DELIMITER ',' CSV HEADER;

-- Copiando os dados do dataset para VENDEDORES
COPY olist.sellers FROM 'C:\Users\kastr\OneDrive\Documents\TCC\olist_sellers_dataset.csv' DELIMITER ',' CSV HEADER;

-- Copiando os dados do dataset para PRODUTOS
COPY olist.products FROM 'C:\Users\kastr\OneDrive\Documents\TCC\olist_products_dataset.csv' DELIMITER ',' CSV HEADER;

-- Copiando os dados do dataset para TRADUÇÃO DE CATEGORIA DE PRODUTO
COPY olist.product_category_name_translation FROM 'C:\Users\kastr\OneDrive\Documents\TCC\product_category_name_translation.csv' DELIMITER ',' CSV HEADER;

-- Copiando os dados do dataset para PEDIDOS
COPY olist.orders FROM 'C:\Users\kastr\OneDrive\Documents\TCC\olist_orders_dataset.csv' DELIMITER ',' CSV HEADER;

-- Copiando os dados do dataset para ITENS DE PEDIDO
COPY olist.order_items FROM 'C:\Users\kastr\OneDrive\Documents\TCC\olist_order_items_dataset.csv' DELIMITER ',' CSV HEADER;

-- Copiando os dados do dataset para PAGAMENTOS DE PEDIDOS
COPY olist.order_payments FROM 'C:\Users\kastr\OneDrive\Documents\TCC\olist_order_payments_dataset.csv' DELIMITER ',' CSV HEADER;

-- Copiando os dados do dataset para GEOLOCALIZAÇÃO
COPY olist.geolocation FROM 'C:\Users\kastr\OneDrive\Documents\TCC\olist_geolocation_dataset.csv' DELIMITER ',' CSV HEADER;
```

Figura 8 – Comando COPY. Fonte: autoria própria

No caso da tabela de avaliações `order_reviews`, foi criada uma tabela temporária

para importar primeiro todos os registros e, em seguida, efetuar a inserção, evitando a duplicação de **review_id** (Figura 68).

```
-- Copiando os dados do dataset para AVALIAÇÕES DE PEDIDOS(Resolvendo problema de Chave Duplicada)
-- Criação de tabela temporária
CREATE TEMP TABLE temp_reviews AS
SELECT * FROM olist.order_reviews WITH NO DATA;

--Copia dos dados para a tabela temporária
COPY temp_reviews FROM 'C:\Users\kastr\OneDrive\Documents\TCC\olist_order_reviews_dataset.csv'
DELIMITER ',' CSV HEADER;

-- Fazendo o INSERT ignorando conflitos
INSERT INTO olist.order_reviews
SELECT * FROM temp_reviews
ON CONFLICT (review_id) DO NOTHING;

-- Limpando tabela temporária
DROP TABLE temp_reviews;
```

Figura 9 – Tabela temporária do order_reviews. Fonte: autoria própria

4.3 Modelagem Não Relacional e ETL no MongoDB

Primeiramente foi realizada a configuração do ambiente do MongoDB, incluindo a instalação do servidor local, criação do banco de dados **brazilian_ecommerce** e definição das coleções que receberiam os dados transformados. Em seguida, foi feita a importação das bibliotecas **pandas** para leitura e manipulação dos arquivos CSV, **datetime** para tratamento de datas, e **MongoClient**, da biblioteca **pymongo**, para a conexão com o MongoDB e execução das operações de inserção (Figura 69).

```
import pandas as pd
from datetime import datetime
from pymongo import MongoClient, ASCENDING
import math
import sys

# ----- CONFIGURAÇÃO -----
MONGO_URI = "mongodb://localhost:27017"
DB_NAME = "brazilian_ecommerce"
BATCH_SIZE = 5000
```

Figura 10 – Configuração do ambiente . Fonte: autoria própria

Após isto, foi utilizada uma função **datetime** para garantir que todos os valores de data e hora que fossem carregados pelo **pandas** fossem convertidos em um formato compatível com o driver do MongoDB (Figura 70).

```
# Função segura para converter pandas Timestamp em datetime do Python ou None
def to_datetime(v):
    if pd.isna(v):
        return None
    if isinstance(v, pd.Timestamp):
        return v.to_pydatetime()
    if isinstance(v, datetime):
        return v
    return None
```

Figura 11 – Função do `datetime`. Fonte: autoria própria

Em seguida, para garantir a integridade e a consistência dos dados, cada arquivo CSV foi carregado no ambiente Python utilizando a biblioteca `pandas`. Foi aplicado o parâmetro `parse_dates` em todas as colunas de marcação temporal (por exemplo, `order_purchase_timestamp`, `order_approved_at`, `order_delivered_customer_date` etc.), assegurando que elas fossem interpretadas como objetos `datetime` em vez de strings (Figura 71) e também foram utilizados mapas auxiliares para mapeamento de geolocalização para anexar latitude e longitude a clientes e vendedores sem repetir buscas.

```
# 2) Carregamento dos CSVs
customers_df = pd.read_csv("olist_customers_dataset.csv")
sellers_df = pd.read_csv("olist_sellers_dataset.csv")
products_df = pd.read_csv("olist_products_dataset.csv")
cat_df = pd.read_csv("product_category_name_translation.csv")
orders_df = pd.read_csv(
    "olist_orders_dataset.csv",
    parse_dates=[
        "order_purchase_timestamp",
        "order_approved_at",
        "order_delivered_carrier_date",
        "order_delivered_customer_date",
        "order_estimated_delivery_date"
    ]
)
items_df = pd.read_csv(
    "olist_order_items_dataset.csv",
    parse_dates=["shipping_limit_date"]
)
payments_df = pd.read_csv("olist_order_payments_dataset.csv")
reviews_df = pd.read_csv(
    "olist_order_reviews_dataset.csv",
    parse_dates=["review_creation_date", "review_answer_timestamp"]
)
geo_df = pd.read_csv("olist_geolocation_dataset.csv")
```

Figura 12 – Comando `parse_dates`. Fonte: autoria própria

Após isto, foi feita a modelagem dos dados com a montagem das coleções através de dicionários e arrays aninhados e gravação em lote com `pymongo.insert_many()` como na Figura 72.

```
# 4) Popula customers
cust_docs = []
for _, r in customers_df.iterrows():
    geo = geo_map.get(r.customer_zip_code_prefix, {})
    cust_docs.append({
        "_id": r.customer_id,
        "unique_id": r.customer_unique_id,
        "zip_code_prefix": int(r.customer_zip_code_prefix),
        "city": r.customer_city,
        "state": r.customer_state,
        "geolocation": {
            "lat": geo.get("geolocation_lat"),
            "lng": geo.get("geolocation_lng")
        }
    })
db.customers.delete_many({})
db.customers.insert_many(cust_docs)
```

Figura 13 – Montagem do documento. Fonte: autoria própria

No modelo não relacional, foi utilizada a opção por explorar três modelagens (modelo normalizado com nove coleções, modelo semi-desnormalizado com duas coleções e modelo totalmente desnormalizado com uma coleção) teve como objetivo avaliar consistência, custo de armazenamento, complexidade de ETL e desempenho de consultas analíticas. Na modelagem com nove coleções as vantagens são menor consumo de armazenamento quando comparado à desnormalização, atualizações pontuais (correção de um produto ou vendedor), maior integridade lógica e menos inconsistências. Já as desvantagens incluem o fato de que as consultas analíticas exigem múltiplos `$lookup` o que aumenta latência e custo computacional em agregações extensas, complexidade maior no pipeline de ETL e menor desempenho para consultas que exigem leitura de muitos campos relacionados. Na modelagem com duas coleções as vantagens são redução na quantidade de `$lookup`, leitura mais rápida do que no modelo totalmente normalizado e menos casos de duplicação. Já as desvantagens são necessidade de `$lookup` para algumas análises, atualizações parciais e complexidade intermediária no ETL. Já na modelagem com uma coleção as vantagens são excelente desempenho em leituras analíticas, simplicidade das consultas. Já as desvantagens são alta duplicação de dados e atualizações de atributos compartilhados (pode ocorrer redundância).

4.3.1 Modelagem com Nove Coleções

A primeira estratégia consistiu em reproduzir o esquema relacional original de nove tabelas, criando nove coleções `customers`, `sellers`, `products`, `product_category_name_translation`, `orders`, `order_items`, `order_payments`, `order_reviews` e `geolocation` (Figura 73).

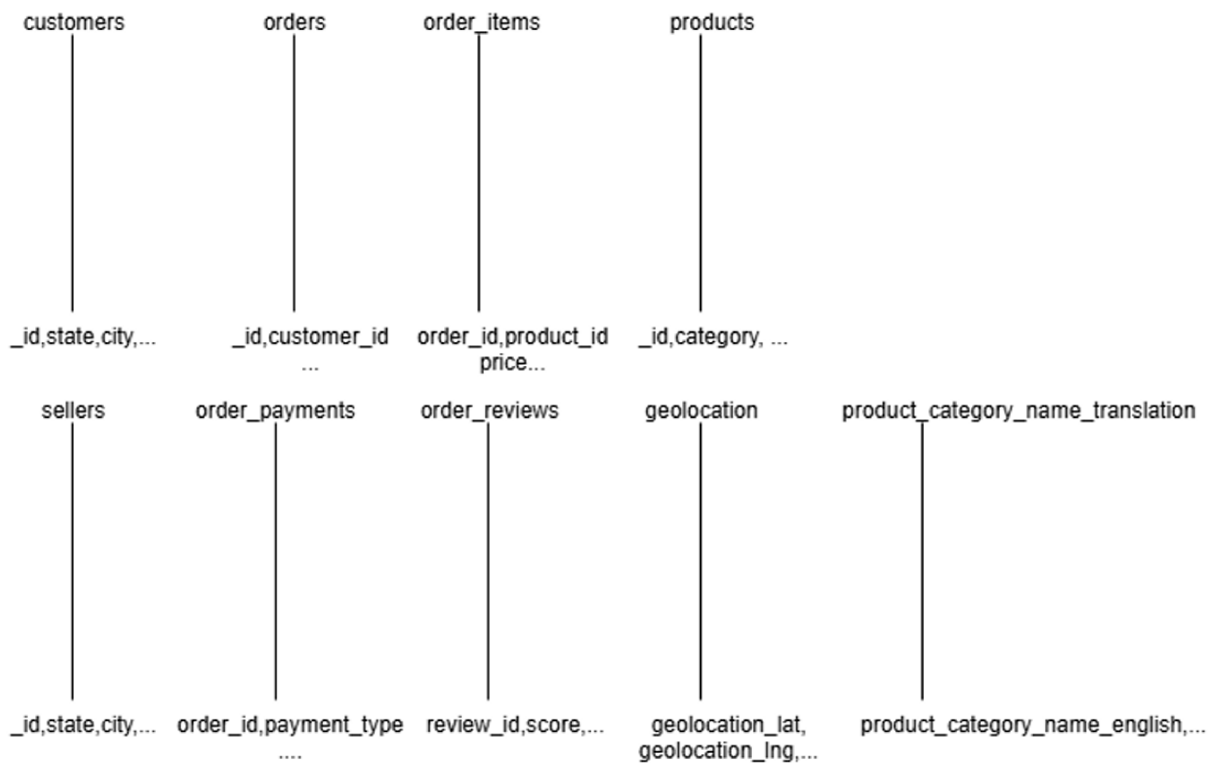


Figura 14 – Modelo com nove coleções. Fonte: autoria própria

Nesse cenário, as consultas agregadas exigiam estágios de agregação com múltiplos `$lookup` entre coleções.

4.3.2 Modelagem com Duas Coleções

Em seguida, foi adotado um modelo intermediário de apenas duas coleções principais: `products`, que possui todos os atributos de produtos, e `orders`, que armazena cada pedido com campos embutidos para cliente, itens (referenciando `product_id` e dados do vendedor), pagamentos e avaliação (Figura 74).

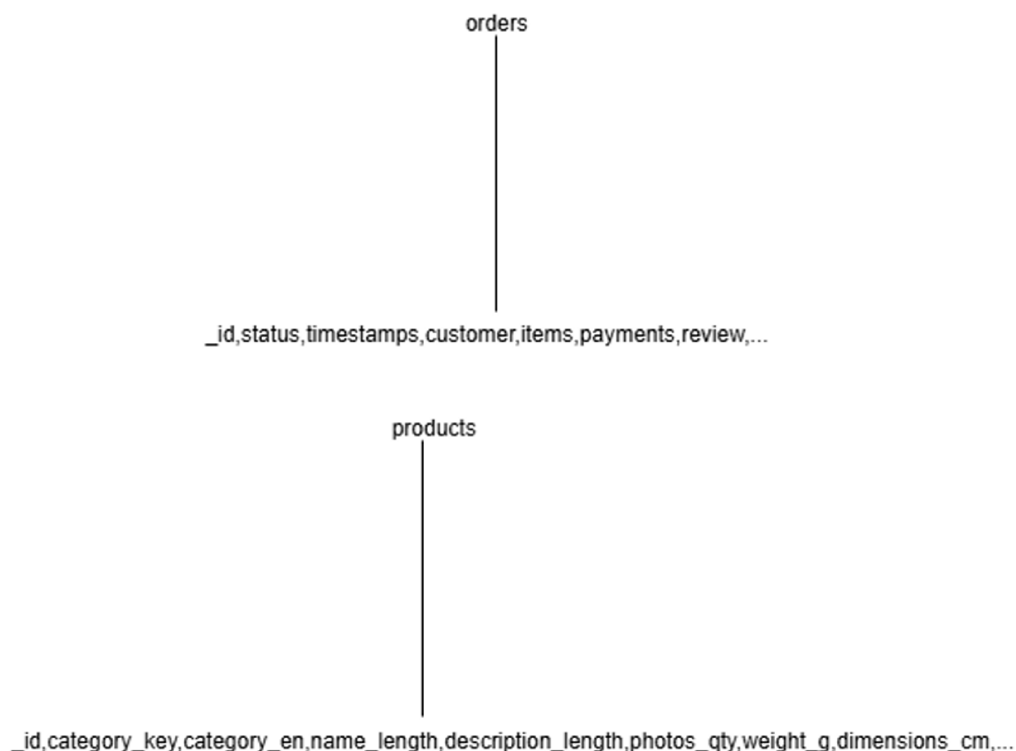


Figura 15 – Modelo com duas coleções. Fonte: autoria própria

Nesse cenário, muitos dados passaram a ficar embutidos ao documento de pedido, mantendo porém a coleção de produtos separada para evitar redundância total.

4.3.3 Modelagem com Uma Coleção

Por fim, optou-se pelo modelo máximo de desnormalização, com uma única coleção **orders** onde cada documento incorpora integralmente todas as informações do pedido: dados do cliente (incluindo geolocalização), lista completa de itens com detalhes do produto e do vendedor, pagamentos e avaliação (Figura 75).

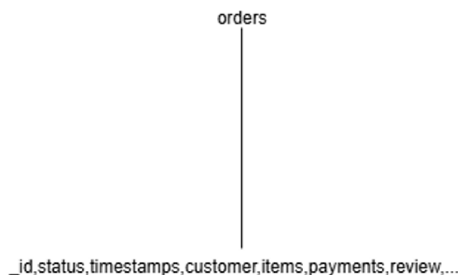


Figura 16 – Modelo com uma coleção. Fonte: autoria própria

Neste cenário, praticamente não há junções e mostrou ser o mais eficiente em leituras analíticas, ainda que exija maior espaço de armazenamento.

4.4 Desenvolvimento das Consultas Agregadas

Nesta etapa, foram elaboradas dez consultas agregadas que representam indicadores de desempenho (KPIs) do e-commerce, e implementadas tanto em SQL no PostgreSQL quanto em pipelines de agregação no MongoDB. O objetivo foi garantir que, independentemente do modelo (relacional ou orientado a documentos), os resultados obtidos pudessem ser similares, demonstrando a equivalência lógica entre as duas abordagens.

4.4.1 Total de Vendas por Estado do Cliente

A primeira consulta calcula o total de vendas por estado do cliente. Para realizar essa consulta no modelo relacional, é necessário acessar as informações de três tabelas: `order_items`, `orders` e `customers`. A especificação da consulta em SQL pode ser vista na Figura 76. Essa consulta seleciona o estado do cliente e calcula a soma dos preços de todos os itens vendidos por estado `SUM(oi.price)`. Para tanto, é necessário fazer uma junção entre as tabelas `order_items`, `orders` e `customers` para relacionar cada item de pedido ao seu respectivo pedido e os pedidos aos seus respectivos clientes, para obter o estado de cada cliente. E por fim, é necessário agrupar os dados por estado dos clientes. Para permitir uma melhor visualização do resultado, os estados foram ordenados do maior para o menor volume de vendas.

```
-- 1) Total de vendas por estado do cliente
SELECT c.customer_state,
       SUM(oi.price) AS total_vendas
FROM order_items oi
JOIN orders o ON oi.order_id = o.order_id
JOIN customers c ON o.customer_id = c.customer_id
GROUP BY c.customer_state
ORDER BY total_vendas DESC;
```

Figura 17 – Consulta Agregada no PostgreSQL. Fonte: autoria própria

No MongoDB para a modelagem com nove coleções, é necessário acessar as informações de três coleções: `order_items`, `orders` e `customers`. A especificação da consulta em MongoDB pode ser vista na Figura 77. O processo inicia na coleção `order_items`, que contém os itens de cada pedido. Primeiro, é feita uma junção `$lookup` com `orders` para associar cada item ao seu respectivo pedido. Em seguida, realiza outra junção com `customers` para relacionar o pedido ao cliente que o realizou e, assim, obter o estado `customer_state` de cada um. Com isto, os dados são agrupados pelo estado do cliente, soma o valor total dos itens vendidos em cada estado. Em seguida, o resultado é projetado para exibir apenas o estado e o total de vendas, e os estados são ordenados de forma decrescente pelo volume de vendas.

```
db.order_items.aggregate([
  { $lookup: {
    from: "orders",
    localField: "order_id",
    foreignField: "_id",
    as: "order"
  }},
  { $unwind: "$order" },
  { $lookup: {
    from: "customers",
    localField: "order.customer_id",
    foreignField: "_id",
    as: "customer"
  }},
  { $unwind: "$customer" },
  { $group: {
    _id: "$customer.customer_state",
    total_vendas: { $sum: "$price" }
  }},
  { $project: { _id: 0, state: "$_id", total_vendas: 1 }},
  { $sort: { total_vendas: -1 } }
]);
```

Figura 18 – Pipeline de Agregação no MongoDB com 9 coleções. Fonte: autoria própria

No MongoDB, para a modelagem com duas coleções, é necessário acessar a coleção `orders`. A especificação da consulta em MongoDB pode ser vista na Figura 78. O processo inicia na coleção `orders`, aplicando `$unwind` sobre o campo `items` para trabalhar cada item individualmente. Em seguida, os dados são agrupados pelo estado do cliente `customer_state`, somando o valor total dos itens vendidos. Por fim, o resultado é ordenado de forma decrescente pelo volume de vendas.

```
db.orders.aggregate([
  { $unwind: "$items" },
  { $group: {
    _id: "$customer.state",
    total_vendas: { $sum: "$items.price" }
  }},
  { $sort: { total_vendas: -1 } }
]);
```

Figura 19 – Pipeline de Agregação no MongoDB com duas coleções. Fonte: autoria própria

No MongoDB, para a modelagem com uma coleção, todas as informações do pedido, dos itens e do cliente já estão armazenadas em `orders`. A especificação da consulta em MongoDB pode ser vista na Figura 79. A pipeline inicia com `$unwind` sobre o campo `items` para tratar cada item separadamente. Em seguida, os dados são agrupados pelo estado do cliente `customer_state`, somando o valor total dos itens vendidos. O resultado é então ordenado do maior para o menor volume de vendas.

```
db.orders.aggregate([
  { $unwind: "$items" },
  { $group: {
    _id: "$customer.state",
    total_vendas: { $sum: "$items.price" }
  }},
  { $sort: { total_vendas: -1 } }
]);
```

Figura 20 – Pipeline de Agregação no MongoDB com uma coleção. Fonte: autoria própria

4.4.2 Produtos Mais Vendidos (Top 10)

A segunda consulta identifica as categorias de produtos mais vendidas (Top 10). Para realizar essa consulta no modelo relacional, é necessário acessar as informações de duas tabelas: `order_items` e `products`. A especificação da consulta em SQL pode ser vista na Figura 80. Essa consulta seleciona o nome da categoria do produto com `product_category_name` e contabiliza quantas vezes os produtos de cada categoria foram vendidos por meio da função `COUNT(*)`. Para tanto, é feita uma junção entre as tabelas `order_items` e `products`, de forma a relacionar cada item de pedido ao seu respectivo produto. Em seguida, é necessário agrupar os dados por categoria de produto, ordenar as categorias das mais vendidas para as menos vendidas e, por fim, limitar o resultado às 10 primeiras categorias utilizando o comando `LIMIT 10`.

```
-- 2) Produtos mais vendidos
SELECT p.product_category_name,
       COUNT(*) AS quantidade_vendas
FROM order_items oi
JOIN products p ON oi.product_id = p.product_id
GROUP BY p.product_category_name
ORDER BY quantidade_vendas DESC
LIMIT 10;
```

Figura 21 – Consulta Agregada no PostgreSQL. Fonte: autoria própria

No MongoDB para a modelagem com nove coleções, é necessário acessar as informações das coleções `order_items` e `products` e `product_category_name_translation`. A especificação da consulta em MongoDB pode ser vista na Figura 81. O processo inicia na coleção `order_items`, que contém os itens de cada pedido. Primeiro, realiza um `$lookup` com a coleção `products` para associar cada item ao produto correspondente, obtendo assim a categoria original. Em seguida, é feito um novo `$lookup` com `product_category_name_translation` para recuperar o nome traduzido da categoria. Depois, os dados são agrupados `$group` por chave da categoria, contando o total de vendas por categoria. Por fim, os resultados são ordenados de forma decrescente pelo número de vendas e limitada a 10 categorias mais vendidas.

```
db.order_items.aggregate([
  { $lookup: {
    from: "products",
    localField: "product_id",
    foreignField: "_id",
    as: "product"
  }},
  { $unwind: "$product" },
  { $lookup: {
    from: "product_category_name_translation",
    localField: "product.product_category_name",
    foreignField: "_id",
    as: "cat"
  }},
  { $unwind: { path: "$cat", preserveNullAndEmptyArrays: true } },
  { $group: {
    _id: { key: "$product.product_category_name", en: "$cat.english" },
    quantidade_vendas: { $sum: 1 }
  }},
  { $sort: { quantidade_vendas: -1 } },
  { $limit: 10 },
  { $project: {
    _id: 0,
    categoria_pt: "$_id.key",
    quantidade_vendas: 1
  }}
]);
```

Figura 22 – Pipeline de Agregação no MongoDB com nove coleções. Fonte: autoria própria

No MongoDB, para a modelagem com duas coleções, é necessário acessar **orders** e **products**. A especificação da consulta em MongoDB pode ser vista na Figura 82. O processo inicia na coleção **orders**, aplicando **\$unwind** sobre o campo **items** para trabalhar cada item individualmente. Em seguida realiza um **\$lookup** para a coleção **products** para associar cada item ao produto correspondente. Depois os dados são agrupados com **\$group** por **product.category_key**, contabilizando vendas por categoria com **\$sum:1**. Por fim o resultado é ordenado de forma decrescente por quantidade de vendas limitado aos 10 primeiros.

```
db.orders.aggregate([
  { $unwind: "$items" },
  { $lookup: {
    from: "products",
    localField: "items.product_id",
    foreignField: "_id",
    as: "product"
  }},
  { $unwind: "$product" },
  { $group: {
    _id: {
      key: "$product.category_key",
      en: "$product.category_en"
    },
    quantidade_vendas: { $sum: 1 }
  }},
  { $sort: { quantidade_vendas: -1 } },
  { $limit: 10 },
  { $project: {
    _id: 0,
    categoria_pt: "$_id.key",
    quantidade_vendas: 1
  }}
]);
```

Figura 23 – Pipeline de Agregação no MongoDB com duas coleções. Fonte: autoria própria

No MongoDB para a modelagem com uma coleção, todas as informações já estão armazenadas em **orders**. A especificação da consulta em MongoDB pode ser vista na Figura 83. O pipeline correspondente inicia com **\$unwind** em **items** permitindo tratar cada item de pedido separadamente, conta quantas vezes cada **product.id** aparece nos itens, somando 1 por ocorrência, ordena o produto com maior número de vendas para o menor, limita apenas os 10 produtos com maior contagem e reformata o documento de saída, removendo **_id** e exibindo **product_id** e **quantidade_vendas**.

```
db.orders.aggregate([
  { $unwind: "$items" },
  { $group: {
    _id: {
      key: "$items.product.category_key",
      en: "$items.product.category_en"
    },
    quantidade_vendas: { $sum: 1 }
  }},
  { $sort: { quantidade_vendas: -1 } },
  { $limit: 10 },
  { $project: {
    _id: 0,
    categoria_pt: "$_id.key",
    quantidade_vendas: 1
  }}
]);
```

Figura 24 – Pipeline de Agregação no MongoDB com uma coleção. Fonte: autoria própria

4.4.3 Receita Total por Categoria de Produto

A terceira consulta calcula a receita total por categoria de produto. Para realizar essa consulta no modelo relacional, é necessário acessar as informações de três tabelas: `order_items`, `products` e `product_category_name_translation`. A especificação da consulta em SQL pode ser vista na Figura 84. Essa consulta seleciona o nome da categoria do produto por meio de `product_category_name` e calcula a soma dos preços de todos os itens daquela categoria com `SUM(oi.price)`. Para tanto, é feita uma junção entre as tabelas `order_items` e `products` para associar cada item ao seu respectivo produto, e outra junção entre `products` e `product_category_name_translation` para obter o nome da categoria em português. Por fim, os resultados são agrupados por categoria de produto e ordenados em ordem decrescente de receita.

```
-- 3) Receita total por categoria de produto
SELECT pct.product_category_name AS categoria,
       SUM(oi.price) AS receita_total
FROM order_items oi
JOIN products p ON oi.product_id = p.product_id
JOIN product_category_name_translation pct
  ON p.product_category_name = pct.product_category_name
GROUP BY categoria
ORDER BY receita_total DESC;
```

Figura 25 – Consulta Agregada no PostgreSQL. Fonte: autoria própria

No MongoDB para a modelagem com nove coleções, é necessário acessar as coleções `order_items`, `products` e `product_category_name_translation`. A especificação da consulta em MongoDB pode ser vista na Figura 85. O processo inicia em `order_items`,

que contém os preços unitários dos itens. Primeiramente, realiza-se um `$lookup` para associar cada item ao seu produto na coleção `products`, obtendo assim a categoria original. Em seguida, um `$group` para somar o valor total por categoria. Depois, realiza um novo `$lookup` com `product_category_name_translation` para obter a tradução do nome da categoria. Por fim, o resultado é projetado para exibir a categoria e a receita total, ordenando os dados de forma decrescente pelo valor.

```
db.order_items.aggregate([
  { $lookup: {
    from: "products",
    localField: "product_id",
    foreignField: "_id",
    as: "product"
  }},
  { $unwind: "$product" },
  { $group: {
    _id: "$product.product_category_name",
    receita_total: { $sum: "$price" }
  }},
  { $lookup: {
    from: "product_category_name_translation",
    localField: "_id",
    foreignField: "_id",
    as: "cat"
  }},
  { $unwind: { path: "$cat", preserveNullAndEmptyArrays: true } },
  { $project: {
    _id: 0,
    categoria_pt: "$_id",
    categoria_en: "$cat.english",
    receita_total: 1
  }},
  { $sort: { receita_total: -1 } }
]);
```

Figura 26 – Pipeline de Agregação no MongoDB com nove coleções. Fonte: autoria própria

No MongoDB, para a modelagem com duas coleções, é necessário acessar `orders` e `products`. A especificação da consulta em MongoDB pode ser vista na Figura 86. O processo inicia na coleção `orders`, aplicando `$unwind` sobre o campo `items` para trabalhar cada item individualmente. Em seguida realiza um `$lookup` para a coleção `products`, para associar cada item ao seu produto. Depois os dados são agrupados com `$group` por `product.category_key`, somando o preço dos itens com `$sum:"$items.price"` para obter a receita total por categoria. Por fim o resultado é projetado com `$project` para exibir a categoria e a receita total, e ordenados de forma decrescente pelo valor.

```
db.orders.aggregate([
  { $unwind: "$items" },
  { $lookup: {
    from: "products",
    localField: "items.product_id",
    foreignField: "_id",
    as: "product"
  }},
  { $unwind: "$product" },
  { $group: {
    _id: "$product.category_key",
    receita_total: { $sum: "$items.price" }
  }},
  { $project: {
    _id: 0,
    categoria_pt: "$_id",
    receita_total: 1
  }},
  { $sort: { receita_total: -1 } }
]);
```

Figura 27 – Pipeline de Agregação no MongoDB com duas coleções. Fonte: autoria própria

No MongoDB para a modelagem com uma coleção, todas as informações já estão armazenadas em `orders`. A especificação da consulta em MongoDB pode ser vista na Figura 87. O pipeline correspondente segue uma linha similar onde inicia com `$unwind` em `items` permitindo tratar cada item de pedido separadamente, agrupa todos esses itens por categoria de produto `items.product.category_key`, para cada categoria, o operador `sum` acumula o valor de `items.price`, que representa o preço de venda de cada item resultando a soma total de vendas associadas a ela.

```
db.orders.aggregate([
  { $unwind: "$items" },
  { $group: {
    _id: "$items.product.category_en",
    receita_total: { $sum: "$items.price" }
  }},
  { $sort: { receita_total: -1 } }
]);
```

Figura 28 – Pipeline de Agregação no MongoDB com uma coleção. Fonte: autoria própria

4.4.4 Top 10 Vendedores por Número de Itens Vendidos

A quarta consulta identifica os vendedores que mais venderam em número de itens. Para realizar essa consulta no modelo relacional, é necessário acessar as informações de duas tabelas: `order_items` e `sellers`. A especificação da consulta em SQL pode ser vista na Figura 88. Essa consulta seleciona o identificador do vendedor e contabiliza o número total de itens vendidos por meio de `COUNT(*)`. Para tanto, é feita uma junção entre as

tabelas `order_items` e `sellers`, relacionando cada item ao respectivo vendedor. Por fim, os dados são agrupados por vendedor, ordenados do que mais vendeu para o que menos vendeu e limitados aos 10 primeiros.

```
-- 4) Top 10 vendedores por número de itens vendidos
SELECT s.seller_id,
       s.seller_city,
       s.seller_state,
       COUNT(*) AS itens_vendidos
FROM order_items oi
JOIN sellers s ON oi.seller_id = s.seller_id
GROUP BY s.seller_id
ORDER BY itens_vendidos DESC
LIMIT 10;
```

Figura 29 – Consulta Agregada no PostgreSQL. Fonte: autoria própria

No MongoDB para a modelagem com nove coleções, é necessário acessar as informações de `order_items` e `sellers`. A especificação da consulta em MongoDB pode ser vista na Figura 89. O processo inicia na coleção `order_items`, agrupando `$group` os registros pelo identificador do vendedor `seller_id` e somando a quantidade de itens vendidos. Em seguida, os resultados são ordenados em ordem decrescente de vendas e limitados aos 10 principais vendedores. Depois, realiza-se um `$lookup` com a coleção `sellers` a fim de obter cidade e estado de cada vendedor. Finalmente, o resultado é projetado para mostrar o identificador do vendedor, cidade, estado e quantidade de itens vendidos.

```
db.order_items.aggregate([
  { $group: {
    _id: "$seller_id",
    itens_vendidos: { $sum: 1 }
  }},
  { $sort: { itens_vendidos: -1 } },
  { $limit: 10 },
  { $lookup: {
    from: "sellers",
    localField: "_id",
    foreignField: "_id",
    as: "seller"
  }},
  { $unwind: { path: "$seller", preserveNullAndEmptyArrays: true } },
  { $project: {
    _id: 0,
    seller_id: "$_id",
    seller_city: "$seller.seller_city",
    seller_state: "$seller.seller_state",
    itens_vendidos: 1
  } }
]);
```

Figura 30 – Pipeline de Agregação no MongoDB com nove coleções. Fonte: autoria própria

No MongoDB, para a modelagem com duas coleções, é necessário acessar **orders** e **products**. A especificação da consulta em MongoDB pode ser vista na Figura 90. O processo inicia na coleção **orders**, aplicando **\$unwind** sobre o campo **items** para trabalhar cada item individualmente. Em seguida realiza-se um **\$group** por **items.seller_id**, contabilizando o número de itens vendidos com **\$sum:1**. No mesmo **\$group** são obtidos com **\$first** os campos **items.seller.city** e **items.seller.state** para apresentar a cidade e o estado do vendedor. Por fim o resultado é ordenado de forma decrescente por itens vendidos, limitado aos 10 primeiros com **\$limit** e projetado com **\$project** para mostrar o identificador do vendedor, cidade, estado e quantidade de itens vendidos.

```
db.orders.aggregate([
  { $unwind: "$items" },
  { $group: {
    _id: "$items.seller_id",
    itens_vendidos: { $sum: 1 },
    seller_city: { $first: "$items.seller.city" },
    seller_state: { $first: "$items.seller.state" }
  }},
  { $sort: { itens_vendidos: -1 } },
  { $limit: 10 },
  { $project: {
    _id: 0,
    seller_id: "$_id",
    seller_city: 1,
    seller_state: 1,
    itens_vendidos: 1
  }}
]);
```

Figura 31 – Pipeline de Agregação no MongoDB com duas coleções. Fonte: autoria própria

No MongoDB para a modelagem com uma coleção, todas as informações já estão armazenadas em **orders**. A especificação da consulta em MongoDB pode ser vista na Figura 91. O pipeline correspondente segue uma linha similar onde inicia com **\$unwind** em **items** permitindo tratar cada item de pedido separadamente, agrupa todos esses item-documentos por ID do vendedor **items.seller.id**, somando 1 para cada item, ordena pelos vendedores com mais itens vendidos e exibe apenas os 10 primeiros.

```

db.orders.aggregate([
  { $unwind: "$items" },
  { $group: {
    _id: "$items.seller.id",
    itens_vendidos: { $sum: 1 },
    cidade: { $first: "$items.seller.city" },
    estado: { $first: "$items.seller.state" }
  }},
  { $project: {
    _id: 0,
    seller_id: "$_id",
    itens_vendidos: 1,
    cidade: 1,
    estado: 1
  }},
  { $sort: { itens_vendidos: -1 } },
  { $limit: 10 }
]);

```

Figura 32 – Pipeline de Agregação no MongoDB com uma coleção. Fonte: autoria própria

4.4.5 Top 5 Categorias mais Vendidas por Número de Pedidos

A quinta consulta apresenta as cinco categorias mais vendidas por número de pedidos. Para realizar essa consulta no modelo relacional, é necessário acessar as informações de três tabelas: `order_items`, `products` e `product_category_name_translation`. A especificação da consulta em SQL pode ser vista na Figura 92. Essa consulta seleciona o nome da categoria e conta quantos pedidos únicos (`order_id`) cada categoria participou por meio de `COUNT(DISTINCT oi.order_id)`. Para tanto, é feita uma junção entre as tabelas `order_items` e `products`, e outra entre `products` e `product_category_name_translation` para obter o nome das categorias em português. Por fim, os resultados são agrupados por categoria, ordenados pelas mais frequentes e limitados às cinco principais.

```

-- 5) Top 5 categorias mais vendidas por número de pedidos
SELECT pct.product_category_name,
       COUNT(DISTINCT oi.order_id) AS total_pedidos
FROM order_items oi
JOIN products p ON oi.product_id = p.product_id
JOIN product_category_name_translation pct
  ON p.product_category_name = pct.product_category_name
GROUP BY pct.product_category_name
ORDER BY total_pedidos DESC
LIMIT 5;

```

Figura 33 – Consulta Agregada no PostgreSQL. Fonte: autoria própria

No MongoDB para a modelagem com nove coleções, é necessário acessar as coleções `order_items` e `products`. A especificação da consulta em MongoDB pode ser vista

na Figura 93. O processo começa em `order_items`, realizando um `$lookup` com `products` para obter a categoria de cada produto. Em seguida, os dados são agrupados por combinação de pedido `order_id` e categoria para garantir que cada pedido seja contado apenas uma vez por categoria. Depois, um segundo `$group` é feito para contar o total de pedidos distintos por categoria. Os resultados são ordenados de forma decrescente pelo número de pedidos e limitados às 5 categorias mais populares. Por fim, o resultado é projetado para exibir apenas o nome da categoria e a quantidade de pedidos.

```
db.order_items.aggregate([
  { $lookup: {
    from: "products",
    localField: "product_id",
    foreignField: "_id",
    as: "product"
  }},
  { $unwind: "$product" },
  { $group: {
    _id: { order_id: "$order_id", categoria: "$product.product_category_name" }
  }},
  { $group: {
    _id: "$_id.categoria",
    total_pedidos: { $sum: 1 }
  }},
  { $sort: { total_pedidos: -1 } },
  { $limit: 5 },
  { $project: {
    _id: 0,
    categoria: "$_id",
    total_pedidos: 1
  } }
]);
```

Figura 34 – Pipeline de Agregação no MongoDB com nove coleções. Fonte: autoria própria

No MongoDB, para a modelagem com duas coleções, é necessário acessar `orders` e `products`. A especificação da consulta em MongoDB pode ser vista na Figura 94. O processo inicia na coleção `orders`, aplicando `$unwind` sobre o campo `items` para trabalhar cada item individualmente. Em seguida realiza um `$lookup` para a coleção `products`, para obter a categoria de cada produto. O primeiro estágio de `$group` agrupa por pedido e categoria. Em seguida um segundo `$group` agrupa por categoria e soma com `$sum:1` para obter o total de pedidos. Por fim o resultado é ordenado de forma decrescente por total de pedidos, limitado aos 5 primeiros com `$limit` e projetado com `$project` para exibir apenas o nome da categoria e a quantidade de pedidos.

```
db.orders.aggregate([
  { $unwind: "$items" },
  { $lookup: {
    from: "products",
    localField: "items.product_id",
    foreignField: "_id",
    as: "product"
  }},
  { $unwind: "$product" },
  { $group: {
    _id: { order_id: "$_id", categoria: "$product.category_key" }
  }},
  { $group: {
    _id: "$_id.categoria",
    total_pedidos: { $sum: 1 }
  }},
  { $sort: { total_pedidos: -1 } },
  { $limit: 5 },
  { $project: {
    _id: 0,
    categoria: "$_id",
    total_pedidos: 1
  } }
]);
```

Figura 35 – Pipeline de Agregação no MongoDB com duas coleções. Fonte: autoria própria

No MongoDB para a modelagem com uma coleção, todas as informações já estão armazenadas em **orders**. A especificação da consulta em MongoDB pode ser vista na Figura 95. O pipeline correspondente segue uma linha similar onde inicia com **\$unwind** em **items** permitindo tratar cada item de pedido separadamente, agrupa todos esses item-documentos com base no ID do pedido e a categoria do produto **items.product.category_key**, depois agrupa novamente por categoria e conta quantos pedidos distintos envolveram cada uma **_id.categoria** e por fim classifica e limita às 5 categorias mais recorrentes em pedidos.

```
db.orders.aggregate([
  { $unwind: "$items" },
  { $group: {
    _id: { order: "$_id", categoria: "$items.product.category_key" }
  }},
  { $group: {
    _id: "$_id.categoria",
    total_pedidos: { $sum: 1 }
  }},
  { $sort: { total_pedidos: -1 } },
  { $limit: 5 }
]);
```

Figura 36 – Pipeline de Agregação no MongoDB com uma coleção. Fonte: autoria própria

4.4.6 Análise de Estados com Maior Número de Pedidos em Parcelas Acima de 12 Vezes

A sexta consulta mapeia os estados onde residem clientes com pedidos contendo mais de nove parcelas. Para realizar essa consulta no modelo relacional, é necessário acessar as informações de três tabelas: `customers`, `orders` e `order_payments`. A especificação da consulta em SQL pode ser vista na Figura 96. Nela, é selecionado o estado do cliente, conta os pedidos únicos. Para tanto, é feita uma junção entre as tabelas `orders` e `customers`, e outra entre `orders` e `order_payments`, filtrando apenas os pagamentos com mais de doze parcelas. Por fim, os resultados são agrupados por estado e número de parcelas, e ordenados de forma decrescente segundo o maior número de parcelas.

```
-- 6) Análise de Estados com Maior Número de Pedidos em Parcelas Acima de 12 Vezes
SELECT c.customer_state AS state,
       COUNT(DISTINCT o.order_id) AS num_orders
FROM orders o
JOIN order_payments op ON o.order_id = op.order_id
JOIN customers c ON o.customer_id = c.customer_id
WHERE op.payment_installments > 12
GROUP BY c.customer_state
ORDER BY num_orders DESC;
```

Figura 37 – Consulta Agregada no PostgreSQL. Fonte: autoria própria

No MongoDB para a modelagem com nove coleções, é necessário acessar as coleções `order_payments`, `orders` e `customers`. A especificação da consulta em MongoDB pode ser vista na Figura 97. O processo inicia na coleção `order_payments`, filtrando `$match` apenas os registros com número de parcelas `payment_installments` maior que doze. Em seguida, os dados são agrupados para obter pedidos únicos. Um `$lookup` com a coleção `orders` é realizado para obter os dados do pedido, e depois outro `$lookup` com `customers` para identificar o estado de cada cliente. Finalmente, agrupa por estado, contando quantos pedidos distintos existem em cada um, projetando o estado e a contagem, e ordenando de forma decrescente pelo número de pedidos.

```
db.order_payments.aggregate([
  { $match: { payment_installments: { $gt: 12 } } },
  { $group: { _id: "$order_id" } },
  { $lookup: {
    from: "orders",
    localField: "_id",
    foreignField: "_id",
    as: "order"
  }},
  { $unwind: "$order" },
  { $lookup: {
    from: "customers",
    localField: "order.customer_id",
    foreignField: "_id",
    as: "customer"
  }},
  { $unwind: "$customer" },
  { $group: {
    _id: "$customer.customer_state",
    num_orders: { $sum: 1 }
  }},
  { $project: {
    _id: 0,
    state: "$_id",
    num_orders: 1
  }},
  { $sort: { num_orders: -1 } }
]);
```

Figura 38 – Pipeline de Agregação no MongoDB com nove coleções. Fonte: autoria própria

No MongoDB, para a modelagem com duas coleções, é necessário acessar **orders** e **products**. A especificação da consulta em MongoDB pode ser vista na Figura 98. O processo inicia na coleção **orders**, aplicando **\$unwind** sobre o campo **payments** para separar cada pagamento em um documento separado. Em seguida aplica **\$match** filtrando pagamentos cuja quantidade de parcelas (**payments.installments**) seja maior que doze. Depois os documentos são agrupados com **\$group** por **customer.state**, usando **\$sum: 1** para contabilizar o número de ocorrências por estado. Por fim o resultado é projetado com **\$project** o estado e a contagem, e ordenando de forma decrescente pelo número de pedidos.

```
db.orders.aggregate([
  { $match: { "payments.installments": { $gt: 12 } } },
  { $group: {
    _id: "$customer.state",
    num_orders: { $sum: 1 }
  }},
  { $project: {
    _id: 0,
    state: "$_id",
    num_orders: 1
  }},
  { $sort: { num_orders: -1, state: 1 } }
]);
```

Figura 39 – Pipeline de Agregação no MongoDB com duas coleções. Fonte: autoria própria

No MongoDB para a modelagem com uma coleção, todas as informações já estão armazenadas em **orders**. A especificação da consulta em MongoDB pode ser vista na Figura 99. O pipeline correspondente segue uma linha similar onde inicia com **\$unwind** em **payments** permitindo tratar cada pagamento de pedido separadamente e filtra os que foram parcelados em mais de doze vezes **payments.installments**, agrupa todos esses item-documentos com base no estado e número de parcelas, armazenando um conjunto de pedidos distintos, depois agrupa novamente apenas por estado e soma os registros com mais de doze parcelas e por fim ordena os estados que mais tiveram ocorrências.

```
db.orders.aggregate([
  { $match: { "payments.installments": { $gt: 12 } } },
  { $group: {
    _id: "$customer.state",
    num_orders: { $sum: 1 }
  }},
  { $project: {
    _id: 0,
    state: "$_id",
    num_orders: 1
  }},
  { $sort: { num_orders: -1 } }
]);
```

Figura 40 – Pipeline de Agregação no MongoDB com uma coleção. Fonte: autoria própria

4.4.7 Pedidos Pagos em Parcela Única

A sétima consulta contabiliza a quantidade de pedidos pagos em parcela única. Para realizar essa consulta no modelo relacional, é necessário acessar as informações de quatro tabelas: **orders**, **order_payments**, **order_items** e **products**. A especificação da consulta em SQL pode ser vista na Figura 100. Nela, é criada uma CTE denominada **pedidos_com_uma_parcela**, que seleciona o nome da categoria e conta quantos pedidos

únicos foram pagos em uma única parcela. Para tanto, é feita uma junção entre as tabelas `orders` e `order_payments`, outra entre `order_items` e `order_payments`, e ainda uma junção entre `order_items` e `products` para obter as informações de categoria dos produtos. Por fim, os resultados são ordenados das categorias mais pedidas para as menos.

```
-- 7) Pedidos pagos em parcela única
WITH pedidos_com_uma_parcela AS (
  SELECT pr.product_category_name,
         COUNT(DISTINCT o.order_id) AS pedidos
  FROM orders o
  JOIN order_payments op ON op.order_id = o.order_id
  JOIN order_items oi ON oi.order_id = op.order_id
  JOIN products pr ON pr.product_id = oi.product_id
  WHERE op.payment_installments = 1
  GROUP BY pr.product_category_name
)
SELECT product_category_name AS categoria,
       pedidos
FROM pedidos_com_uma_parcela
ORDER BY pedidos DESC,
       product_category_name DESC;
```

Figura 41 – Consulta Agregada no PostgreSQL. Fonte: autoria própria

No MongoDB para a modelagem com nove coleções, é necessário acessar as coleções `order_items` e `products` e `order_payments`. A especificação da consulta em MongoDB pode ser vista na Figura 101. A consulta começa na coleção `order_payments` para identificar os pedidos que tiveram pelo menos um pagamento. Primeiro é feito um `$match` para filtrar esses pagamentos e em seguida um `$group` que reduz para pedidos únicos. Depois, realiza um `$lookup` de `order_items` para trazer os itens daqueles pedidos. Em seguida um `$lookup` em `products` recupera o `product_category_name` de cada item. Por fim, os dados são agrupados por categoria e o resultado é projetado com o categoria e o total de pedidos e ordenado de forma decrescente pelo total.

```
db.order_payments.aggregate([
  { $match: { payment_installments: 1 } },
  { $group: { _id: "$order_id" } },
  { $lookup: {
    from: "order_items",
    localField: "_id",
    foreignField: "order_id",
    as: "items"
  }},
  { $unwind: "$items" },
  { $lookup: {
    from: "products",
    localField: "items.product_id",
    foreignField: "_id",
    as: "product"
  }},
  { $unwind: "$product" },
  { $group: {
    _id: "$product.product_category_name",
    pedidos_set: { $addToSet: "$_id" }
  }},
  { $project: {
    categoria: "$_id",
    total_pedidos: { $size: "$pedidos_set" }
  }},
  { $sort: { total_pedidos: -1 } }
]);
```

Figura 42 – Pipeline de Agregação no MongoDB com nove coleções. Fonte: autoria própria

No MongoDB, para a modelagem com duas coleções, é necessário acessar **orders** e **products**. A especificação da consulta em MongoDB pode ser vista na Figura 102. O processo inicia na coleção **orders**, aplicando **\$unwind** sobre o campo **payments** para trabalhar cada pagamento individualmente. Em seguida aplica **\$match** para filtrar apenas os pedidos pagos em parcela única. Depois utiliza **\$unwind** sobre **items** para processar cada item de pedido separadamente. Realiza então um **\$lookup** para a coleção **products**, para recuperar o **product_category_name** de cada item. Na etapa seguinte, com **\$group**, os documentos são agrupados por **product.category_key**, utilizando **\$addToSet** para criar um conjunto único de **_id** de pedidos. Por fim, os dados são agrupados por categoria e o resultado é projetado com o categoria e o total de pedidos e ordenado de forma decrescente pelo total.

```
db.orders.aggregate([
  { $unwind: "$payments" },
  { $match: { "payments.installments": 1 } },
  { $unwind: "$items" },
  { $lookup: {
    from: "products",
    localField: "items.product_id",
    foreignField: "_id",
    as: "product"
  }},
  { $unwind: "$product" },
  { $group: {
    _id: "$product.category_key",
    pedidos_set: { $addToSet: "$_id" }
  }},
  { $project: {
    _id: 0,
    categoria: "$_id",
    total_pedidos: { $size: "$pedidos_set" }
  }},
  { $sort: { total_pedidos: -1 } }
]);
```

Figura 43 – Pipeline de Agregação no MongoDB com duas coleções. Fonte: autoria própria

No MongoDB para a modelagem com uma coleção, todas as informações já estão armazenadas em `orders`. A especificação da consulta em MongoDB pode ser vista na Figura 103. O pipeline correspondente segue uma linha similar onde inicia com `$unwind` em `payments` permitindo tratar cada pagamento de pedido separadamente e filtra os pagamentos que foram feitos à vista `payments.installments`, depois novamente `$unwind` em `items` permitindo tratar cada item do pedido separadamente, Agrupa por ID do produto e adiciona o ID do pedido a um conjunto, renomeia os campos e calcula o tamanho do conjunto de pedidos e por fim classifica os produtos mais vendidos em pedidos pagos à vista.(Figura 103).

```
db.orders.aggregate([
  { $match: { "payments.installments": 1 } },
  { $unwind: "$items" },
  { $group: {
    _id: "$items.product.category_key",
    pedidos_set: { $addToSet: "$_id" }
  }},
  { $project: {
    _id: 0,
    tipo_pagamento: { $literal: "parcela_unica" },
    product_category_name: { $ifNull: ["$_id", "UNKNOWN"] },
    pedidos: { $size: "$pedidos_set" }
  }},
  { $sort: { pedidos: -1, product_category_name: -1 } }
]);
```

Figura 44 – Pipeline de Agregação no MongoDB com uma coleção. Fonte: autoria própria

4.4.8 Número de Pedidos Bem-Sucedidos e Valor Total por Método de Pagamento

A oitava consulta calcula o número de pedidos bem-sucedidos e o valor total por método de pagamento. Para realizar essa consulta no modelo relacional, é necessário acessar as informações de duas tabelas: `order_payments` e `orders`. A especificação da consulta em SQL pode ser vista na Figura 104. Essa consulta seleciona o tipo de pagamento, conta os pedidos distintos e soma os valores pagos, considerando apenas os pedidos bem-sucedidos. Para tanto, é feita uma junção entre as tabelas `orders` e `order_payments` para relacionar o pedido ao método de pagamento. Por fim, os resultados são agrupados por método de pagamento, apresentando o número de pedidos e o valor total, e ordenados em ordem alfabética decrescente.

```
-- 8) Número de pedidos bem-sucedidos e o valor total do pagamento com base no método
    de pagamento
SELECT op.payment_type AS metodo,
       COUNT(DISTINCT o.order_id) AS total_pedidos,
       SUM(CASE
            WHEN o.order_status NOT IN ('cancelled', 'unavailable')
            THEN op.payment_value
            END) AS pagamento_total
FROM order_payments op
JOIN orders o ON o.order_id = op.order_id
GROUP BY op.payment_type
ORDER BY op.payment_type DESC;
```

Figura 45 – Consulta Agregada no PostgreSQL. Fonte: autoria própria

No MongoDB para a modelagem com nove coleções, é necessário acessar `order_payments` e `orders`. A especificação da consulta em MongoDB pode ser vista na Figura 105. A consulta começa na coleção `order_payments` para agregar por `payment_type`. Primeiro há um `$lookup` em `orders` para obter o status do pedido. Após isto, desnormaliza a coleção `orders`, agrupa por `payment_type`, acumulando pedidos distintos `$addToSet` e somando os valores, onde pedidos cancelados são ignorados. O resultado é projetado com método de pagamento, total de pedidos e valor total, ordenado alfabeticamente pelo método.

```
db.order_payments.aggregate([
  { $lookup: {
    from: "orders",
    localField: "order_id",
    foreignField: "_id",
    as: "order"
  }},
  { $unwind: "$order" },
  { $group: {
    _id: "$payment_type",
    pedidos_set: { $addToSet: "$order_id" },
    pagamento_total: {
      $sum: {
        $cond: [
          { $in: [ "$order.order_status", ["cancelled", "unavailable"] ] },
          0,
          "$payment_value"
        ]
      }
    }
  }},
  { $project: {
    metodo: "$_id",
    total_pedidos: { $size: "$pedidos_set" },
    pagamento_total: 1
  }},
  { $sort: { metodo: -1 } }
]);
```

Figura 46 – Pipeline de Agregação no MongoDB com nove coleções. Fonte: autoria própria

No MongoDB, para a modelagem com duas coleções, é necessário acessar a coleção `orders`. A especificação da consulta em MongoDB pode ser vista na Figura 106. O processo inicia aplicando `$unwind` sobre o campo `payments` para trabalhar cada pagamento individualmente. Em seguida, com `$group`, os documentos são agrupados pelo tipo de pagamento (`payments.type`). Dentro do agrupamento, o pagamento total é calculado. Simultaneamente, utiliza `$addToSet` para criar um conjunto contendo os `_id` dos pedidos, garantindo que cada pedido seja contado apenas uma vez por método de pagamento. Depois, com `$project` é projetado com método de pagamento, total de pedidos e valor total, ordenado alfabeticamente pelo método.

```
db.orders.aggregate([
  { $unwind: "$payments" },
  { $group: {
    _id: "$payments.type",
    pedidos_set: { $addToSet: "$_id" },
    pagamento_total: {
      $sum: {
        $cond: [
          { $in: [ "$status", ["cancelled", "unavailable"] ] },
          0, // Ignora pedidos cancelados
          "$payments.value"
        ]
      }
    }
  }
}],
{ $project: {
  _id: 0,
  metodo: "$_id",
  total_pedidos: { $size: "$pedidos_set" },
  pagamento_total: 1
}},
{ $sort: { metodo: -1 } }
]);
```

Figura 47 – Pipeline de Agregação no MongoDB com duas coleções. Fonte: autoria própria

No MongoDB para a modelagem com uma coleção, todas as informações já estão armazenadas em **orders**. A especificação da consulta em MongoDB pode ser vista na Figura 107. O pipeline correspondente segue uma linha similar onde inicia com **\$unwind** em **payments** permitindo tratar cada pagamento de pedido separadamente e agrupa por tipo de pagamento, coletando os pedidos únicos, depois soma o valor de pagamento apenas se o pedido não foi cancelado ou indisponível, apresenta o método, total de pedidos e valor total e por fim ordena alfabeticamente inverso.

```
db.orders.aggregate([
  { $unwind: "$payments" },
  { $group: {
    _id: "$payments.type",
    pedidos_set: { $addToSet: "$_id" },
    pagamento: {
      $sum: {
        $cond: [
          { $in: [ "$status", ["cancelled", "unavailable"] ] },
          0,
          "$payments.value"
        ]
      }
    }
  }
}],
{ $project: {
  _id: 0,
  metodo: "$_id",
  total_pedidos: { $size: "$pedidos_set" },
  pagamento: 1
}},
{ $sort: { metodo: -1 } }
]);
```

Figura 48 – Pipeline de Agregação no MongoDB com uma coleção. Fonte: autoria própria

4.4.9 Receita e Quantidade de Pedidos por Estado e Método de Pagamento

A penúltima consulta retorna a receita e a quantidade de pedidos por estado e método de pagamento. Para realizar essa consulta no modelo relacional, é necessário acessar as informações de três tabelas: `orders`, `customers` e `order_payments`. A especificação da consulta em SQL pode ser vista na Figura 108. Essa consulta seleciona o estado do cliente, o tipo de pagamento, o número total de pedidos únicos e a soma do valor pago. Para tanto, é feita uma junção entre as tabelas `orders` e `customers`, e outra entre `orders` e `order_payments`, obtendo o estado e o método de pagamento. Por fim, os resultados são agrupados por estado e método de pagamento, ordenados em ordem decrescente de receita.

```
-- 9) Receita e quantidade de pedidos por estado e método de pagamento
SELECT c.customer_state AS estado,
       op.payment_type AS metodo_pagamento,
       COUNT(DISTINCT o.order_id) AS total_pedidos,
       SUM(oi.receita_pedido) AS receita_total
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id
JOIN (
    SELECT order_id,
           SUM(price) AS receita_pedido
    FROM order_items
    GROUP BY order_id
) oi ON o.order_id = oi.order_id
JOIN order_payments op ON o.order_id = op.order_id
GROUP BY c.customer_state, op.payment_type
ORDER BY c.customer_state ASC, receita_total DESC;
```

Figura 49 – Consulta Agregada no PostgreSQL. Fonte: autoria própria

No MongoDB para a modelagem com nove coleções, é necessário acessar as coleções `customers`, `orders` e `order_payments`. A especificação da consulta em MongoDB pode ser vista na Figura 109 e Figura 110. A consulta começa na coleção `orders`. Primeiro, junta `order_items` para calcular a receita por pedido. Em seguida, realiza dois `$lookup` com `order_payments` para métodos de pagamento e com `customers` para obter o estado. Desnormaliza os arrays de pagamento e cliente. Agrupa por combinação estado e método, somando receita e acumulando pedidos distintos. Projeta estado, método, receita total e contagem de pedidos, ordenando por estado e receita decrescente.

```
db.orders.aggregate([
  { $lookup: {
    from: "order_items",
    localField: "_id",
    foreignField: "order_id",
    as: "items"
  }},
  { $addFields: {
    order_revenue: { $sum: "$items.price" }
  }},
  { $lookup: {
    from: "order_payments",
    localField: "_id",
    foreignField: "order_id",
    as: "payments"
  }},
  { $lookup: {
    from: "customers",
    localField: "customer_id",
    foreignField: "_id",
    as: "customer"
  }},
  { $unwind: "$customer" },
  { $unwind: "$payments" },
  { $group: {
    _id: {
      estado: "$customer.customer_state",
      metodo_pagamento: "$payments.payment_type"
    },
    receita_total: { $sum: "$order_revenue" },
    pedidos: { $addToSet: "$_id" }
  }},
  { $project: {
    _id: 0,
    estado: "$_id.estado",
    metodo_pagamento: "$_id.metodo_pagamento",
    receita_total: 1,
    total_pedidos: { $size: "$pedidos" }
  }},
  { $sort: { estado: 1, receita_total: -1 } }
]);
```

Figura 50 – Pipeline de Agregação no MongoDB com nove coleções. Fonte: autoria própria

No MongoDB, para a modelagem com duas coleções, é necessário acessar a coleção `orders`. A especificação da consulta em MongoDB pode ser vista na Figura 111. O processo inicia aplicando `$unwind` sobre o campo `payments` para trabalhar cada pagamento individualmente. Em seguida, realiza um `$group` agrupando pelo estado do cliente (`customer.state`) e pelo método de pagamento (`payments.type`). Nesse agrupamento, a receita total é calculada somando os valores de `payments.value`, e o conjunto `pedidos_set` é formado com `$addToSet` contendo os `_id` dos pedidos, garantindo que cada pedido seja contado apenas uma vez por combinação de estado e método de pagamento. Depois, com `$project` projeta o estado, método, receita total e contagem de

pedidos, ordenando por estado e receita decrescente.

```
db.orders.aggregate([
  { $unwind: "$payments" },
  { $group: {
    _id: {
      estado: "$customer.state",
      metodo_pagamento: "$payments.type"
    },
    receita_total: { $sum: "$payments.value" },
    pedidos_set: { $addToSet: "$_id" }
  }},
  { $project: {
    _id: 0,
    estado: "$_id.estado",
    metodo_pagamento: "$_id.metodo_pagamento",
    receita_total: 1,
    total_pedidos: { $size: "$pedidos_set" }
  }},
  { $sort: { estado: 1, receita_total: -1 } }
]);
```

Figura 51 – Pipeline de Agregação no MongoDB com duas coleções. Fonte: autoria própria

No MongoDB para a modelagem com uma coleção, todas as informações já estão armazenadas em **orders**. A especificação da consulta em MongoDB pode ser vista na Figura 112. O pipeline correspondente segue uma linha similar onde inicia com **\$unwind** em **payments** permitindo tratar cada pagamento de pedido separadamente, e depois novamente **\$unwind** em **items** permitindo tratar cada item do pedido separadamente. Agrupa por estado e tipo de pagamento, soma a receita e coleta os pedidos únicos com **\$addToSet** mais **\$size**. E por fim, apresenta os dados por estado e método, ordenando pela maior receita dentro de cada estado. Figura 112.

```
db.orders_full.aggregate([
  { $project: {
    _id: 1,
    "customer.state": 1,
    payments: 1
  }},
  { $unwind: "$payments" },
  { $group: {
    _id: {
      estado: "$customer.state",
      metodo_pagamento: "$payments.type"
    },
    receita_total: { $sum: "$payments.value" },
    pedidos_set: { $addToSet: "$_id" }
  }},
  { $project: {
    _id: 0,
    estado: "$_id.estado",
    metodo_pagamento: "$_id.metodo_pagamento",
    receita_total: { $round: ["$receita_total", 2] },
    total_pedidos: { $size: "$pedidos_set" }
  }},
  { $sort: { estado: 1, receita_total: -1 } }
]);
```

Figura 52 – Pipeline de Agregação no MongoDB com uma coleção. Fonte: autoria própria

4.4.10 Top 5 Categorias por Receita Total, Tempo Médio de Entrega, Número de Itens e Número de Pedidos

Por fim, a décima consulta apresenta as cinco categorias de produto com maior receita e seu tempo médio de entrega. Para realizar essa consulta no modelo relacional, é necessário acessar as informações de quatro tabelas: `order_items`, `orders`, `products` e `product_category_name_translation`. A especificação da consulta em SQL pode ser vista na Figura 113. Essa consulta seleciona a categoria, soma a receita e calcula o tempo médio de entrega em dias, utilizando a função `EPOCH` para converter a diferença de datas em segundos e posteriormente em dias. Para tanto, é feita uma junção entre as tabelas `orders` e `order_items`, outra entre `order_items` e `products`, e outra entre `products` e `product_category_name_translation` para obter a categoria em português. Por fim, os resultados são agrupados por categoria, ordenados pela maior receita e limitados às cinco principais.

```
-- 10) Top 5 categorias por receita total, tempo médio de entregas (em dias),  
-- número de itens e número de pedidos  
SELECT p.product_category_name AS categoria,  
       SUM(oi.price) AS receita_total,  
       ROUND(AVG(EXTRACT(EPOCH FROM (o.order_delivered_customer_date -  
                                   o.order_purchase_timestamp)) / 86400)::numeric, 2)  
       AS tempo_medio_entrega,  
       COUNT(*) AS numero_itens,  
       COUNT(DISTINCT o.order_id) AS numero_pedidos  
FROM order_items oi  
JOIN products p ON oi.product_id = p.product_id  
JOIN orders o ON oi.order_id = o.order_id  
WHERE o.order_delivered_customer_date IS NOT NULL  
GROUP BY p.product_category_name  
ORDER BY receita_total DESC  
LIMIT 5;
```

Figura 53 – Consulta Agregada no PostgreSQL. Fonte: autoria própria

No MongoDB para a modelagem com nove coleções, é necessário acessar as coleções `products`, `orders` e `order_items`. A especificação da consulta em MongoDB pode ser vista na Figura 114 e Figura 115. A consulta inicia na coleção `order_items`. Primeiro junta `products` para obter categorias e `orders` para datas de entrega. Filtra pedidos entregues e calcula dias de entrega. Agrupa por categoria, somando receita, calculando média de dias de entrega `$avg` e acumulando pedidos distintos. Projeta categoria, receita, média de dias, total de itens e pedidos, ordenando por receita decrescente e limitando ao top 5.

```
db.order_items.aggregate([
  { $lookup: {
    from: "products",
    localField: "product_id",
    foreignField: "_id",
    as: "product"
  }},
  { $unwind: "$product" },
  { $lookup: {
    from: "orders",
    localField: "order_id",
    foreignField: "_id",
    as: "order"
  }},
  { $unwind: "$order" },
  { $match: {
    "order.order_delivered_customer_date": { $ne: null }
  }},
  { $addFields: {
    delivery_days: {
      $divide: [
        { $subtract: [ "$order.order_delivered_customer_date",
          "$order.order_purchase_timestamp" ] },
        1000 * 60 * 60 * 24
      ]
    }
  }},
  { $group: {
    _id: "$product.product_category_name",
    receita_total: { $sum: "$price" },
    tempo_medio_entrega: { $avg: "$delivery_days" },
    numero_itens: { $sum: 1 },
    pedidos: { $addToSet: "$order_id" }
  }},
  { $project: {
    _id: 0,
    categoria: "$_id",
    receita_total: 1,
    tempo_medio_entrega: { $round: ["$tempo_medio_entrega", 2] },
    numero_itens: 1,
    numero_pedidos: { $size: "$pedidos" }
  }},
  { $sort: { receita_total: -1 } },
  { $limit: 5 }
]);
```

Figura 54 – Pipeline de Agregação no MongoDB com nove coleções. Fonte: autoria própria

No MongoDB, para a modelagem com duas coleções, é necessário acessar **orders** e **products**. A especificação da consulta em MongoDB pode ser vista na Figura 116 e Figura 117. O processo inicia na coleção **orders** filtrando apenas os pedidos que possuem data de entrega registrada (**timestamps.delivered_customer** por meio de **\$match**. Em seguida, aplica **\$unwind** sobre o campo **items** para processar cada item de pedido individualmente. Depois realiza um **\$lookup** com a coleção **products**, para obter categorias.

Com `$addFields`, calcula o número de dias de entrega (`delivery_days`) dividindo a diferença entre as datas `delivered_customer` e `purchase` por 86.400 (milissegundos em um dia). Em seguida, os dados são agrupados (`$group`) por `product.category_key`, somando o preço dos itens para obter a receita total e calculando a média de dias de entrega com `$avg`. Com `$project`, projeta a categoria, receita, média de dias, total de itens e pedidos, ordenando por receita decrescente e limitando ao top 5.

```
db.orders.aggregate([
  { $match: { "timestamps.delivered_customer": { $ne: null } } },
  { $unwind: "$items" },
  { $lookup: {
    from: "products",
    localField: "items.product_id",
    foreignField: "_id",
    as: "product"
  }},
  { $unwind: "$product" },
  { $addFields: {
    dias_entrega: {
      $divide: [
        { $subtract: ["$timestamps.delivered_customer",
                     "$timestamps.purchase"] },
        86400000
      ]
    }
  }},
  { $addFields: {
    item_revenue: {
      $cond: [
        { $gt: [ { $ifNull: ["$items.quantity", 0] }, 0 ] },
        { $multiply: ["$items.price", "$items.quantity"] },
        "$items.price"
      ]
    }
  }},
  { $group: {
    _id: "$product.category_key",
    receita_total: { $sum: "$item_revenue" },
    media_dias_entrega: { $avg: "$dias_entrega" }
  }},
  { $project: {
    _id: 0,
    categoria: "$_id",
    receita_total: { $round: ["$receita_total", 2] },
    media_dias_entrega: { $round: ["$media_dias_entrega", 2] }
  }},
  { $sort: { receita_total: -1 } },
  { $limit: 5 }
]);
```

Figura 55 – Pipeline de Agregação no MongoDB com duas coleções. Fonte: autoria própria

No MongoDB para a modelagem com uma coleção, todas as informações já estão armazenadas em `orders`. A especificação da consulta em MongoDB pode ser vista na

Figura 118. O pipeline correspondente segue uma linha similar onde inicia com `$unwind` em `items` permitindo tratar cada item do pedido separadamente. Cria os campos categoria, receita e total de entregas. Agrupa por categoria, somando a receita total, média de dias de entrega e número de vendas. E por fim, ordena por maior receita e limita a cinco categorias e formata a saída com duas casas decimais na média de entrega e exibe os campos categoria, receita total e tempo médio de entrega em dias.

```
db.orders.aggregate([
  { $match: { "timestamps.delivered_customer": { $ne: null } } },
  { $unwind: "$items" },
  { $addFields: {
    dias_entrega: {
      $divide: [
        { $subtract: ["$timestamps.delivered_customer",
                     "$timestamps.purchase"] },
        86400000
      ]
    }
  }},
  { $addFields: {
    item_revenue: {
      $cond: [
        { $gt: [ { $ifNull: ["$items.quantity", 0] }, 0 ] },
        { $multiply: ["$items.price", "$items.quantity"] },
        "$items.price"
      ]
    }
  }},
  { $group: {
    _id: "$items.product.category_key",
    receita_total: { $sum: "$item_revenue" },
    tempo_medio_entrega_dias: { $avg: "$dias_entrega" },
    categoria_en: { $first: "$items.product.category_en" }
  }},
  { $project: {
    _id: 0,
    categoria: "$_id",
    receita_total: { $round: ["$receita_total", 2] },
    tempo_medio_entrega_dias: { $round: ["$tempo_medio_entrega_dias", 2] }
  }},
  { $sort: { receita_total: -1 } },
  { $limit: 5 }
]);
```

Figura 56 – Pipeline de Agregação no MongoDB com uma coleção. Fonte: autoria própria

5 Resultados e Discussão

Neste capítulo, primeiramente, são apresentados os resultados das consultas analíticas implementadas tanto no PostgreSQL quanto no MongoDB, comparando para cada indicador os valores obtidos e confirmando a similaridade dos resultados. Em seguida, são mostradas as métricas de desempenho coletadas (tempo de execução médio). Dessa maneira, foi possível discutir na seção de Comparação dos Resultados as vantagens e desvantagens de cada abordagem no âmbito de um cenário real de e-commerce e visualizar também o impacto da desnormalização nas leituras analíticas e o custo de manutenção de índices em junções, oferecendo recomendações práticas para a escolha do modelo mais adequado a diferentes perfis de carga de trabalho. Por fim, na seção de Apêndices é feita a descrição de todos os resultados das dez consultas analíticas implementadas tanto no PostgreSQL quanto no MongoDB.

5.1 Resultado das Consultas Seleccionadas Para Comparação

Para documentar os resultados das consultas analíticas seleccionadas, foram elaboradas as Figuras 57, 58, 59, 60, 61, 62, 63, nas quais se encontram os valores retornados por cada consulta. A Figura 57 e 58, referente à consulta de “Total de Vendas por Estado do Cliente” (Veja Seção 4.4.1), apresenta o valor total de vendas para cada estado, ordenado de forma decrescente. A Figura 59 e 60, referente à consulta de “Top 5 Categorias mais Vendidas por Número de Pedidos” (Veja Seção 4.4.5), apresenta as cinco categorias que mais foram vendidas, ordenado de forma decrescente. A Figura 61, 62 e 63, referente à consulta de “Pedidos Pagos em Parcela Única” (Veja Seção 4.4.7), apresenta a quantidade total de pedidos efetuados em parcela única para cada categoria, também ordenado de forma decrescente.

	customer_state text	total_vendas numeric
1	SP	5202955.05
2	RJ	1824092.67
3	MG	1585308.03
4	RS	750304.02
5	PR	683083.76
6	SC	520553.34
7	BA	511349.99
8	DF	302603.94
9	GO	294591.95
10	ES	275037.31
11	PE	262788.03
12	CE	227254.71
13	PA	178947.81
14	MT	156453.53
15	MA	119648.22
16	MS	116812.64
17	PB	115268.08
18	PI	86914.08
19	RN	83034.98
20	AL	80314.81
21	SE	58920.85
22	TO	49621.74
23	RO	46140.64
24	AM	22356.84
25	AC	15982.95
26	AP	13474.30
27	RR	7829.43

Figura 57 – Total de Vendas por Estado do Cliente.
Fonte: autoria própria

```
< {
  total_vendas: 5202955.05,
  state: 'SP'
}
{
  total_vendas: 1824092.67,
  state: 'RJ'
}
{
  total_vendas: 1585308.03,
  state: 'MG'
}
{
  total_vendas: 750304.02,
  state: 'RS'
}
{
  total_vendas: 683083.76,
  state: 'PR'
}
{
  total_vendas: 520553.34,
  state: 'SC'
}
{
  total_vendas: 511349.99,
  state: 'BA'
}
{
  total_vendas: 302603.94,
  state: 'DF'
}
{
  total_vendas: 294591.95,
  state: 'GO'
}
}
```

Figura 58 – Produtos Mais Vendidos (Top 10). Fonte: autoria própria

	product_category_name [PK] text	total_pedidos bigint
1	cama_mesa_banho	9417
2	beleza_saude	8836
3	esporte_lazer	7720
4	informatica_acessorios	6689
5	moveis_decoracao	6449

Figura 59 – Top 5 Categorias mais Vendidas por Número de Pedidos. Fonte: autoria própria

```
< {  
  total_pedidos: 9417,  
  categoria: 'cama_mesa_banho'  
}  
{  
  total_pedidos: 8836,  
  categoria: 'beleza_saude'  
}  
{  
  total_pedidos: 7720,  
  categoria: 'esporte_lazer'  
}  
{  
  total_pedidos: 6689,  
  categoria: 'informatica_acessorios'  
}  
{  
  total_pedidos: 6449,  
  categoria: 'moveis_decoracao'  
}
```

Figura 60 – Top 5 Categorias mais Vendidas por Número de Pedidos. Fonte: autoria própria

	tipo_pagamento text	product_category_name text	pedidos bigint
1	parcela_unica	esporte_lazer	4299
2	parcela_unica	informatica_acessorios	4177
3	parcela_unica	beleza_saude	3880
4	parcela_unica	cama_mesa_banho	3535
5	parcela_unica	moveis_decoracao	3152
6	parcela_unica	utilidades_domesticas	2732
7	parcela_unica	telefonia	2552
8	parcela_unica	eletronicos	2052
9	parcela_unica	automotivo	2041
10	parcela_unica	brinquedos	1902
11	parcela_unica	relorios_presentes	1890
12	parcela_unica	ferramentas_jardim	1814
13	parcela_unica	cool_stuff	1451
14	parcela_unica	bebes	1386
15	parcela_unica	perfumaria	1238
16	parcela_unica	papelaria	1153
17	parcela_unica	fashion_bolsas_e_acessorios	999
18	parcela_unica	pet_shop	852
19	parcela_unica	[null]	741
20	parcela_unica	eletrodomesticos	585
21	parcela_unica	consoles_games	580
22	parcela_unica	moveis_escritorio	550
23	parcela_unica	malas_acessorios	414
24	parcela_unica	construcao_ferramentas_construcao	351
25	parcela_unica	instrumentos_musicais	315
26	parcela_unica	livros_interesse_geral	312
27	parcela_unica	alimentos	290
28	parcela_unica	eletroportateis	281
29	parcela_unica	bebidas	229
30	parcela_unica	casa_construcao	203
31	parcela_unica	audio	195
32	parcela_unica	moveis_sala	184
33	parcela_unica	livros_tecnicos	159

Figura 61 – Pedidos Pagos em Parcela Única. Fonte: autoria própria

```
{
  categoria: 'brinquedos',
  total_pedidos: 1902
}
{
  categoria: 'relorios_presentes',
  total_pedidos: 1890
}
}
{
  categoria: 'ferramentas_jardim',
  total_pedidos: 1814
}
}
{
  categoria: 'cool_stuff',
  total_pedidos: 1451
}
}
{
  categoria: 'bebes',
  total_pedidos: 1386
}
}
{
  categoria: 'perfumaria',
  total_pedidos: 1238
}
}
{
  categoria: 'papelaria',
  total_pedidos: 1153
}
}
{
  categoria: 'fashion_bolsas_e_acessorios',
  total_pedidos: 999
}
}
{
  categoria: 'pet_shop',
  total_pedidos: 852
}
}
```

Figura 62 – Pedidos Pagos em Parcela Única. Fonte: autoria própria

```
< {
  categoria: 'esporte_lazer',
  total_pedidos: 4299
}
{
  categoria: 'informatica_acessorios',
  total_pedidos: 4177
}
}
{
  categoria: 'beleza_saude',
  total_pedidos: 3880
}
}
{
  categoria: 'cama_mesa_banho',
  total_pedidos: 3535
}
}
{
  categoria: 'moveis_decoracao',
  total_pedidos: 3152
}
}
{
  categoria: 'utilidades_domesticas',
  total_pedidos: 2732
}
}
{
  categoria: 'telefonia',
  total_pedidos: 2552
}
}
{
  categoria: 'eletronicos',
  total_pedidos: 2052
}
}
{
  categoria: 'automotivo',
  total_pedidos: 2041
}
}
```

Figura 63 – Pedidos Pagos em Parcela Única. Fonte: autoria própria

5.2 Comparação dos Resultados

As consultas implementadas produziram resultados semanticamente equivalentes nos SGBDs PostgreSQL e MongoDB como pode ser visto no exemplo na Figura 64 e Figura 65. Essa equivalência foi alcançada ao se aplicar as mesmas regras de contagem e alocação de receita em ambos os sistemas. A comparabilidade foi garantida mediante o uso de parâmetros de filtro idênticos em todas as consultas.

	categoria text	receita_total numeric	tempo_medio_entrega_dias numeric
1	beleza_saude	1233211.61	11.98052502919498139776
2	relogios_presentes	1165898.98	12.66318005046193538612
3	cama_mesa_banho	1023434.76	12.80646297860217562603
4	esporte_lazer	954695.05	12.15205680809798055694
5	informatica_acessorios	888613.62	13.22530387433914353980

Figura 64 – Resultado da última consulta no PostgreSQL

```
< {
  receita_total: 1233211.61,
  categoria: 'beleza_saude',
  tempo_medio_entrega: 11.98
}
{
  receita_total: 1165898.98,
  categoria: 'relogios_presentes',
  tempo_medio_entrega: 12.66
}
{
  receita_total: 1023434.76,
  categoria: 'cama_mesa_banho',
  tempo_medio_entrega: 12.81
}
{
  receita_total: 954695.05,
  categoria: 'esporte_lazer',
  tempo_medio_entrega: 12.15
}
{
  receita_total: 888613.62,
  categoria: 'informatica_acessorios',
  tempo_medio_entrega: 13.23
}
```

Figura 65 – Resultado da última consulta no MongoDB

5.3 Análise Interpretativa

Os resultados identificaram padrões de comportamento relevantes para o domínio de e-commerce. Observou-se, por exemplo, concentração de receita em determinados estados (capitais), categorias com alto volume de vendas e categorias com receita média elevada. A presença de vendedores que concentram grande número de itens vendidos pode indicar alta rotatividade. Pedidos com muitas parcelas concentrados em certos estados podem apontar diferenças socioeconômicas, sendo informações úteis para decisões de marketing e logística. Além disso, as métricas de tempo médio de entrega por categoria mostraram que dependendo do produto alguns aspectos podem interferir na agilidade da entrega como fragilidade, necessidade de transporte refrigerado, localização do vendedor em relação ao cliente, etc.

5.4 Validação e Comparação de Desempenho entre PostgreSQL e MongoDB

5.4.1 Ambiente de Teste

Os testes de desempenho foram realizados na seguinte configuração :

- Máquina: Intel(R) Core(TM) i5-10300H, 8GB, 475GB
- Sistema operativo: Microsoft Windows 11
- Versão PostgreSQL: 15.2
- Versão MongoDB: 8.0.10
- Dados: utilização do conjunto de dados `Olist Brazilian E-commerce` contendo aproximadamente 100.000 pedidos (126.19 MB).

5.4.2 Metodologia de Medição

Para obter medidas confiáveis, foi adotado a seguinte metodologia:

1. **Repetições:** cada consulta foi executada dez vezes e foi utilizado a média.
2. **Ferramentas de medição:**
 - PostgreSQL: `EXPLAIN (ANALYZE)` para obter tempo real.
 - MongoDB: `db.collection.explain("executionStats").aggregate([...])` para obter `executionTimeMillis`.
3. **Condições controladas:** rede local (ambos bancos na mesma máquina), e desligamento de serviços não essenciais durante testes.

5.4.3 Tabelas de Comparação

As Tabelas a seguir, possuem as medições feitas para as consultas agregadas tanto no PostgreSQL quanto no MongoDB

Tabela 2 – Tempos médios de execução das consultas.

Seção	PostgreSQL (ms)	MongoDB 9 coleções (ms)	MongoDB 2 coleções (ms)	MongoDB 1 coleção (ms)
4.4.1	178,773	88,480	1,279	1,379
4.4.2	92,634	69,704	32,962	1,470
4.4.3	117,800	41,875	33,749	1,703
4.4.4	86,191	571	1,214	1,531
4.4.5	591,765	42,446	43,834	3,380
4.4.6	93,475	1,587	1,589	1,900
4.4.7	365,011	56,328	19,122	2,940
4.4.8	652,624	34,344	1,516	1,600
4.4.9	921,639	148,318	3,917	4,103
4.4.10	185,584	138,025	42,045	3,129
<i>Observação: Os números das consultas corresponde a seção indicada.</i>				

Os resultados indicam que as dez consultas avaliadas apresentaram equivalência semântica entre PostgreSQL e MongoDB, isto é, produziram as mesmas contagens, somas e médias quando aplicadas sobre modelagens correspondentes, o que valida a comparabilidade dos SGBDs no contexto do estudo. Em termos práticos, os achados reforçam um trade-off de desempenho versus manutenção/consistência. Enquanto o modelo relacional favorece integridade e governança (com `joins` bem indexados e ferramentas de inspeção como `EXPLAIN ANALYZE`), a modelagem documental desnormalizada no MongoDB simplifica leituras agregadas ao custo de maior cuidado com atualizações e possíveis redundâncias.

6 Conclusões e Recomendações para Trabalhos Futuros

Tendo em vista os objetivos deste trabalho, comparar semânticamente as consultas agregadas e avaliar o desempenho entre modelos de dados normalizados e desnormalizados nos SGBDs PostgreSQL e MongoDB, utilizando como base o conjunto de dados públicos de e-commerce brasileiro disponibilizado pela Olist no [Kaggle](#). Conclui-se que foram obtidos resultados satisfatórios. Para as dez consultas propostas, observou-se equivalência nos resultados retornados, o que comprova que as diferenças de modelagem não comprometeram a consistência dos indicadores calculados.

Em termos de desempenho, verificou-se que a modelagem desnormalizada no MongoDB (coleção única) apresentou menores tempos médios de execução para consultas agregadas, especialmente naquelas que, em modelos normalizados, demandariam múltiplos JOINS ou o operador \$lookup. Já a modelagem normalizada, tanto no PostgreSQL quanto no MongoDB com múltiplas coleções, mostrou-se mais eficiente do ponto de vista de manutenção, economia de espaço em disco e flexibilidade para atualizações parciais dos dados, embora com maior custo de processamento para agregações complexas.

Além de atingir os objetivos propostos, o trabalho alinhou-se com a justificativa apresentada ao explorar, de forma sistemática, três graus de agregação distintos (9 coleções, 2 coleções e 1 coleção), permitindo discutir de forma fundamentada o desempenho de leitura, o custo de escrita e a complexidade do processo de ETL. Essa abordagem forneceu evidências que podem apoiar a decisão de arquitetos de dados na escolha da modelagem mais adequada para diferentes cenários de carga de trabalho.

Os resultados obtidos também possibilitaram uma análise interpretativa sobre o comportamento de clientes, vendedores e categorias de produtos no contexto de e-commerce, com métricas como tempo médio de entrega, receita por estado e métodos de pagamento, bem como a identificação de padrões e preferências de consumo. Essas análises demonstraram que, independentemente da tecnologia utilizada, a qualidade do modelo de dados e o cuidado no pré-processamento têm impacto direto na confiabilidade das informações extraídas.

Para esforços futuros, recomenda-se expandir os testes para bases de dados maiores e com maior volume de atualizações concorrentes, a fim de avaliar o impacto da escalabilidade horizontal sobre o desempenho das consultas. Sugere-se também a implementação de mecanismos automáticos de **benchmark**, integrando scripts de medição com coleta de métricas de EXPLAIN e geração de relatórios comparativos.

Adicionalmente, propõe-se investigar o uso de técnicas de pré-agregação, como `materialized views` no PostgreSQL e `pipelines` agendados no MongoDB, para reduzir tempos de resposta em cenários de consultas repetitivas. Tais estudos poderão fornecer auxílio para a construção de um guia prático de escolha de modelagem e tecnologia, considerando não apenas o desempenho bruto, mas também a manutenibilidade e o custo operacional ao longo do ciclo de vida do sistema.

Como contribuição prática, o material (`scripts` de preparação e carga dos dados, `pipelines` MQL) foi disponibilizado em repositório público no [Github](#).

Referências

OLIST. Brazilian E-commerce Public Dataset (Conjunto de Dados). Kaggle, 2018. Disponível em: <<https://www.kaggle.com/datasets/olistbr/brazilian-ecommerce>>. Acesso em: 13 ago. 2025.

GANDOMI, Amir; HAIDER, Murtaza. Beyond the hype: Big data concepts, methods, and analytics. *International Journal of Information Management*, v. 35, n. 2, p. 137–144, 2015. Disponível em: <<https://encurtador.com.br/E5Jos>>. Acesso em: 31 out. 2024.

INMON, William H. *Building the Data Warehouse*. John Wiley Sons, 2005. Disponível em: <<https://encurtador.com.br/7Rx6B>>. Acesso em: 31 out. 2024.

MELO, Felipe F. de L.; GOUVEIA, Roberta M. M.; DE ALENCAR, Andrêza L.; BATISTA, Maria da C. M.; SANTOS NETO, Ademir B.; FERREIRA, Tiago A. E. Performance analysis of NoSQL databases with large volumes of open educational data. *International Journal of Computer Applications*, v. 174, n. 29, p. 9–17, 2021. Disponível em: <<https://ijcaonline.org/archives/volume174/number29/31859-2021921219/>>. Acesso em: 31 out. 2024.

CHODOROW, Kristina. *MongoDB: The Definitive Guide*. O'Reilly Media, 2013. Disponível em: <https://ocw.uc3m.es/pluginfile.php/5943/mod_page/content/40/3_3_mongoDB_OCW.pdf>. Acesso em: 31 out. 2024.

RATHORE, M.; BAGUI, S. S. MongoDB: meeting the dynamic needs of modern applications. *Encyclopedia*, v. 4, n. 4, p. 1433–1453, 2024. Disponível em: <<https://doi.org/10.3390/encyclopedia4040093>>. Acesso em: 31 out. 2024.

POKORNY, Jan. NoSQL databases: a step to database scalability in web environment. *International Journal of Web Information Systems*, v. 16, n. 3, p. 309–327, 2020. Disponível em: <https://www.researchgate.net/publication/221237715_NoSQL_Databases_a_step_to_database_scalability_in_Web_environment>. Acesso em: 31 out. 2024.

MONGODB INC. *MongoDB: Manual and Documentation*. Disponível em: <<https://www.mongodb.com/docs/manual/>>. Acesso em: 31 out. 2024.

DIETER, F.; SEBASTIAN, S.; LUDWIG, S. Data Modeling for NoSQL Document-Oriented Databases. *Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development*, p. 198–205, 2015. Disponível em: <<https://ceur-ws.org/Vol-1478/paper17.pdf>>. Acesso em: 13 ago. 2025.

LOPES, R. et al. A Conceptual Model Based Design Methodology for MongoDB Databases. *Journal of Database Management*, v. 32, n. 3, p. 45–67, 2021. Disponível em: <<https://ieeexplore.ieee.org/document/10541595>>. Acesso em: 13 ago. 2025.

KUMAR, Teerath; ZHANG, Cheng; RAJ, Kislay; ROY, Arunabha M.; LUO, Bin. Performance analysis in NoSQL databases, relational databases and NoSQL databases as a service in the cloud. *Big Data and Cognitive Computing*, v. 7, n. 2, p. 97, 2023. Disponível em: <<https://www.mdpi.com/2504-2289/7/2/97>>. Acesso em: 6 out. 2024.

GYÖRÖDI, Cornelia A.; DUMȘE-BURESCU, Diana V.; ZMARANDA, Doina R.; GYÖRÖDI, Robert Ș. A Comparative Study of MongoDB and Document-Based MySQL for Big Data Application Data Management. *Big Data and Cognitive Computing*, v. 6, n. 2, p. 49, 2022. Disponível em: <<https://www.mdpi.com/2504-2289/6/2/49>>. Acesso em: 6 out. 2024.

MICHELS, Jan; HARE, Keith; KULKARNI, Krishna; ZUZARTE, Calisto; LIU, Zhen Hua; HAMMERSCHMIDT, Beda; ZEMKE, Fred. The new and improved SQL:2016 standard. *SIGMOD Record*, v. 47, n. 2, p. 51–60, 2018. DOI: 10.1145/3299887.3299897. Disponível em: <https://sigmodrecord.org/publications/sigmodRecord/1806/pdfs/08_Industry_Michels.pdf>. Acesso em: 13 ago. 2025.

CHASSEUR, Craig; LI, Yinan; PATEL, Jignesh M. Enabling JSON Document Stores in Relational Systems. *Proceedings of WebDB*, 2013. Disponível em: <<https://pages.cs.wisc.edu/~chasseur/argo-long.pdf>>. Acesso em: 13 ago. 2025.

MISHRA, O.; LODHI, P.; MEHTA, S. Document Oriented NoSQL Databases: An Empirical Study. In: PANDA, B.; SHARMA, S.; ROY, N. (Eds.). *Data Science and Analytics (REDSET 2017)*. Singapore: Springer, 2018. Disponível em: <https://www.academia.edu/90247120/Document_Oriented_NoSQL_Databases_An_Empirical_Study> Acesso em: 12 out. 2024.

HERIPRACOY, Sulistyo. Big Data Analysis with MongoDB for Decision Support System. *TELKOMNIKA*, v. 14, n. 3, p. 1083-1089, 2016. Disponível em: <<https://encurtador.com.br/MVEa2>>. Acesso em: 12 out. 2024.

MOHAN, Rishi Kesav; KANMANI, Risheek Rakshit Sukumar; GANESAN, Krishna Anandan; RAMASUBRAMANIAN, Nisha. Evaluating NoSQL Databases for OLAP Workloads: A Benchmarking Study of MongoDB, Redis, Kudu and ArangoDB. *arXiv preprint arXiv:2405.17731*, 2024. Disponível em: <<https://arxiv.org/abs/2405.17731>>. Acesso em: 13 ago. 2025.

XU, Zhenyu; TAN, Yimin; WANG, Xinxue; LI, Xiang. Data-Driven Analysis for the Operation Status of the e-Commerce Platform based on Olist. *Data-Driven Analysis for the Operation Status of the e-Commerce Platform Based on Olist — Volume 1: ICPDI*,

2022. p. 859–867. SciTePress, 2022. Disponível em: <<https://www.scitepress.org/Papers/2022/118366/>>. Acesso em: 13 ago. 2025.

CODD, E. F. A relational model of data for large shared data banks. *Communications of the ACM*, v. 13, n. 6, p. 377-387, 1970. Disponível em: <<https://dl.acm.org/doi/pdf/10.1145/362384.362685>>. Acesso em: 30 ago. 2025.

ELMASRI, R.; NAVATHE, S. B. *Fundamentals of Database Systems*. 7. ed. Boston: Pearson, 2016. Disponível em: <<https://encurtador.com.br/6buHQ>>. Acesso em: 30 ago. 2025.

GRAY, Jim; REUTER, Andreas. *Transaction Processing: Concepts and Techniques*. San Francisco: Morgan Kaufmann, 1993. Disponível em: <<https://encurtador.com.br/BmI15>>. Acesso em: 30 ago. 2025.

CORBELLINI, A. et al. Persisting Data in the Web: From Relational to NoSQL Databases. In: *WEB INFORMATION SYSTEMS AND TECHNOLOGIES*. Springer, 2017. p. 9-25. Disponível em: <<https://www.sciencedirect.com/science/article/abs/pii/S0306437916303210>>. Acesso em: 30 ago. 2025.

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. *Sistema de Banco de Dados*. 6. ed. [S.l.]: Elsevier, 2020. Disponível em: <<https://integrada.minhabiblioteca.com.br/reader/books/9788595157552/>>. Acesso em: 30 ago. 2025.

CARDOSO, J. *Introdução aos Bancos de Dados*. [S.l.]: FCA, 2012. Disponível em: <<https://integrada.minhabiblioteca.com.br/reader/books/9788502162839/>>. Acesso em: 30 ago. 2025.

HAN, J. et al. Survey on NoSQL database. In: *2011 6TH INTERNATIONAL CONFERENCE ON PERVASIVE COMPUTING AND APPLICATIONS*. [S.l.: s.n.], 2011. p. 363-366. Disponível em: <<https://tarjomefa.com/wp-content/uploads/2015/06/3179-English.pdf>>. Acesso em: 30 ago. 2025.

POSTGRESQL GLOBAL DEVELOPMENT GROUP. *PostgreSQL 15.2 Documentation*. 2025. Disponível em: <<https://www.postgresql.org/docs/current/using-explain.html#USING-EXPLAIN-ANALYZE>>. Acesso em: 30 ago. 2025.

MONGODB INC. *MongoDB Aggregation Framework*. 2025. Disponível em: <<https://www.mongodb.com/docs/manual/aggregation/>>. Acesso em: 30 ago. 2025.

SILVA, O. et al. *Banco de Dados NoSQL e MongoDB*. [S.l.]: Editora XYZ, 2021. Disponível em: <<https://integrada.minhabiblioteca.com.br/reader/books/9786556901534/>>. Acesso em: 30 ago. 2025.

COOPER, B. F. et al. Benchmarking Cloud Serving Systems with YCSB. In: *PROCEEDINGS OF THE 1ST ACM SYMPOSIUM ON CLOUD COMPUTING*. [S.l.: s.n.], 2010. p. 143-154. Disponível em: <<https://courses.cs.duke.edu/fall13/compsci590>>.

[4/838-CloudPapers/ycsb.pdf](#)>. Acesso em: 30 ago. 2025.

CHEN, D. et al. Online Retail Dataset. UCI Machine Learning Repository, 2018. Disponível em: <<https://archive.ics.uci.edu/ml/datasets/Online+Retail>>. Acesso em: 30 ago. 2025.

INSTACART. Instacart Market Basket Analysis Dataset. 2017. Disponível em: <<https://www.kaggle.com/competitions/instacart-market-basket-analysis/data>>. Acesso em: 30 ago. 2025.

RETAIL ROCKET. E-commerce Dataset. 2019. Disponível em: <<https://www.kaggle.com/retailrocket/ecommerce-dataset>>. Acesso em: 30 ago. 2025.

PedroGuarnieri2108. projeto-tcc-olist-mql. GitHub, 2025. Disponível em: <<https://github.com/PedroGuarnieri2108/projeto-tcc-olist-mql>>. Acesso em: 22 set. 2025.

Apêndices

A Resultados Completos das Consultas

A.1 Resultados das Consultas no PostgreSQL

Para documentar os resultados das dez consultas analíticas executadas no PostgreSQL, foram elaboradas as Figuras 63 a 69, nas quais se encontram os valores retornados por cada consulta. A Figura 63, referente à consulta de “Total de Vendas por Estado do cliente” (Veja Seção 4.4.1), apresenta o valor total de vendas para cada estado, ordenado de forma decrescente. Na Figura 64, referente à consulta de “Top 10 Produtos Mais Vendidos” (Veja Seção 4.4.2), apresenta a quantidade de vendas por categoria de produto, ordenado de forma decrescente. Na Figura 65, referente à consulta de “Receita Total por Categoria de Produto” (Veja Seção 4.4.3), apresenta o valor total de vendas para cada categoria de produto, ordenado de forma decrescente. Na Figura 68, referente à consulta de “Top 10 Vendedores por Número de Itens Vendidos” (Veja Seção 4.4.4), apresenta os id dos dez vendedores que mais venderam itens junto com seu estado de origem, cidade e a quantidade de itens vendidos, ordenado de forma decrescente. Na Figura 69, referente à consulta de “Top 5 Categorias mais Vendidas por Número de Pedidos”(Veja Seção 4.4.5), apresenta as Top 5 categorias que mais foram vendidas, ordenado de forma decrescente.

	customer_state text 🔒	total_vendas numeric 🔒
1	SP	5202955.05
2	RJ	1824092.67
3	MG	1585308.03
4	RS	750304.02
5	PR	683083.76
6	SC	520553.34
7	BA	511349.99
8	DF	302603.94
9	GO	294591.95
10	ES	275037.31
11	PE	262788.03
12	CE	227254.71
13	PA	178947.81
14	MT	156453.53
15	MA	119648.22
16	MS	116812.64
17	PB	115268.08
18	PI	86914.08
19	RN	83034.98
20	AL	80314.81
21	SE	58920.85
22	TO	49621.74
23	RO	46140.64
24	AM	22356.84
25	AC	15982.95
26	AP	13474.30
27	RR	7829.43

Figura 66 – Total de Vendas por Estado do Cliente.
Fonte: autoria própria

	product_category_name text 🔒	quantidade_vendas bigint 🔒
1	cama_mesa_banho	11115
2	beleza_saude	9670
3	esporte_lazer	8641
4	moveis_decoracao	8334
5	informatica_acessorios	7827
6	utilidades_domesticas	6964
7	relorios_presentes	5991
8	telefonias	4545
9	ferramentas_jardim	4347
10	automotivo	4235

Figura 67 – Produtos Mais Vendidos (Top 10). Fonte: autoria própria

	categoria text	receita_total numeric
1	beleza_saude	1258681.34
2	relogios_presentes	1205005.68
3	cama_mesa_banho	1036988.68
4	esporte_lazer	988048.97
5	informatica_acessorios	911954.32
6	moveis_decoracao	729762.49
7	cool_stuff	635290.85
8	utilidades_domesticas	632248.66
9	automotivo	592720.11
10	ferramentas_jardim	485256.46
11	brinquedos	483946.60
12	bebes	411764.89
13	perfumaria	399124.87
14	telefonias	323667.53
15	moveis_escritorio	273960.70
16	papelaria	230943.23
17	pcs	222963.13
18	pet_shop	214315.41
19	instrumentos_musicais	191498.88
20	eletroportateis	190648.58
21	eletronicos	160246.74
22	consoles_games	157465.22
23	fashion_bolsas_e_acessorios	152823.54
24	construcao_ferramentas_construcao	144677.59
25	malas_acessorios	140429.98
26	eletrodomesticos_2	113317.74
27	casa_construcao	83088.12
28	eletrodomesticos	80171.53
29	agro_industria_e_comercio	72530.47
30	moveis_sala	68916.56
31	telefonias_fixa	59583.00
32	casa_conforto	58572.04
33	climatizacao	55024.96

Figura 68 – Receita Total por Categoria de Produto - Parte 1. Fonte: autoria própria

67	flores	1110.04
68	casa_conforto_2	760.27
69	cds_dvds_musicais	730.00
70	fashion_roupa_infanto_juvenil	569.85
71	seguros_e_servicos	283.29

Figura 70 – Receita Total por Categoria de Produto - Parte 3. Fonte: autoria própria

34	audio	50688.50
35	portateis_casa_forno_e_cafe	47445.71
36	livros_interesse_geral	46856.88
37	moveis_cozinha_area_de_servico_jantar_e_jardim	46328.37
38	construcao_ferramentas_iluminacao	41080.00
39	construcao_ferramentas_seguranca	40544.52
40	industria_comercio_e_negocios	39669.61
41	alimentos	29393.41
42	market_place	28378.47
43	construcao_ferramentas_jardim	25715.89
44	artes	24202.64
45	fashion_calcados	23562.77
46	bebidas	22428.70
47	sinalizacao_e_seguranca	21509.23
48	moveis_quarto	20028.78
49	livros_tecnicos	19096.06
50	construcao_ferramentas_ferramentas	15903.95
51	alimentos_bebidas	15179.48
52	fashion_roupa_masculina	10797.82
53	fashion_underwear_e_moda_praia	9541.55
54	artigos_de_natal	8800.82
55	tablets_impressao_imagem	7528.41
56	cine_foto	6933.46
57	musica	6034.35
58	dvds_blu_ray	5999.39
59	livros_importados	4639.85
60	artigos_de_festas	4485.18
61	moveis_colchao_e_estofado	4368.08
62	fashion_roupa_feminina	2803.64
63	fashion_esporte	2119.51
64	la_cuisine	2054.99
65	artes_e_artesanato	1814.01
66	fraldas_higiene	1567.59

Figura 69 – Receita Total por Categoria de Produto - Parte 2. Fonte: autoria própria

	seller_id [PK] text	seller_city text	seller_state text	itens_vendidos bigint
1	6560211a19b47992c3666cc44a7e94c0	sao paulo	SP	2033
2	4a3ca9315b744ce9f8e9374361493884	ibitinga	SP	1987
3	1f50f920176fa81dab994f9023523100	sao jose do rio preto	SP	1931
4	cc419e0650a3c5ba77189a1882b7556a	santo andre	SP	1775
5	da8622b14eb17ae2831f4ac5b9dab84a	piracicaba	SP	1551
6	955fee9216a65b617aa5c0531780ce60	sao paulo	SP	1499
7	1025f0e2d44d7041d6cf58b6550e0bfa	sao paulo	SP	1428
8	7c67e1448b00f6e969d365cea6b010ab	itaguaquecetuba	SP	1364
9	ea8482cd71df3c1969d7b9473ff13abc	sao paulo	SP	1203
10	7a67c85e85bb2ce8582c35f2203ad736	sao paulo	SP	1171

Figura 71 – Top 10 Vendedores por Número de Itens Vendidos. Fonte: autoria própria

	product_category_name [PK] text	total_pedidos bigint
1	cama_mesa_banho	9417
2	beleza_saude	8836
3	esporte_lazer	7720
4	informatica_acessorios	6689
5	moveis_decoracao	6449

Figura 72 – Top 5 Categorias mais Vendidas por Número de Pedidos. Fonte: autoria própria

Na sequência, a Figura 70, referente à consulta de “Análise de Estados com Maior Número de Pedidos em Parcelas Acima de 12 Vezes” (Veja Seção 4.4.6), apresenta a quantidade de pedidos feitos por estado com mais de doze parcelas, ordenado de forma decrescente. Na Figura 71, referente à consulta de “Pedidos Pagos em Parcela Única” (Veja Seção 4.4.7), apresenta a quantidade total de pedidos para cada categoria, ordenado de forma decrescente. Na Figura 73, referente à consulta de “Número de Pedidos Bem-Sucedidos e Valor Total por Método de Pagamento” (Veja Seção 4.4.8), apresenta a quantidade de pedidos efetivados por método de pagamento e valor total. Na Figura 74, referente à consulta de “Receita e Quantidade de Pedidos por Estado e Método de Pagamento” (Veja Seção 4.4.9), apresenta o valor total e a quantidade de pedidos feitos por estado para cada método de pagamento e ordenado de forma decrescente. Na Figura 78, referente à consulta de “Top 5 Categorias por Receita e Tempo Médio de Entrega” (Veja Seção 4.4.10), apresenta as cinco melhores categorias que obtiveram mais lucro e a média de entrega.

	state text	num_orders bigint
1	SP	62
2	RJ	26
3	MG	19
4	RS	16
5	BA	13
6	PR	13
7	PE	7
8	CE	5
9	PB	4
10	ES	4
11	GO	3
12	SC	3
13	PA	3
14	DF	2
15	SE	1
16	RO	1
17	PI	1
18	MA	1
19	AL	1

Figura 73 – Análise de Estados com Maior Número de Pedidos em Parcelas Acima de 12 Vezes. Fonte: autoria própria

	tipo_pagamento text	product_category_name text	pedidos bigint
1	parcela_unica	esporte_lazer	4299
2	parcela_unica	informatica_acessorios	4177
3	parcela_unica	beleza_saude	3880
4	parcela_unica	cama_mesa_banho	3535
5	parcela_unica	moveis_decoracao	3152
6	parcela_unica	utilidades_domesticas	2732
7	parcela_unica	telefonia	2552
8	parcela_unica	eletronicos	2052
9	parcela_unica	automotivo	2041
10	parcela_unica	brinquedos	1902
11	parcela_unica	relogios_presentes	1890
12	parcela_unica	ferramentas_jardim	1814
13	parcela_unica	cool_stuff	1451
14	parcela_unica	bebes	1386
15	parcela_unica	perfumaria	1238
16	parcela_unica	papelaria	1153
17	parcela_unica	fashion_bolsas_e_acessorios	999
18	parcela_unica	pet_shop	852
19	parcela_unica	[null]	741
20	parcela_unica	eletrodomesticos	585
21	parcela_unica	consoles_games	580
22	parcela_unica	moveis_escritorio	550
23	parcela_unica	malas_acessorios	414
24	parcela_unica	construcao_ferramentas_construcao	351
25	parcela_unica	instrumentos_musicais	315
26	parcela_unica	livros_interesse_geral	312
27	parcela_unica	alimentos	290
28	parcela_unica	eletroportateis	281
29	parcela_unica	bebidas	229
30	parcela_unica	casa_construcao	203
31	parcela_unica	audio	195
32	parcela_unica	moveis_sala	184
33	parcela_unica	livros_tecnicos	159

Figura 74 – Pedidos Pagos em Parcela Única - Parte 1. Fonte: autoria própria

34	parcela_unica	market_place	155
35	parcela_unica	alimentos_bebidas	154
36	parcela_unica	casa_conforto	140
37	parcela_unica	climatizacao	134
38	parcela_unica	construcao_ferramentas_iluminacao	124
39	parcela_unica	telefonia_fixa	116
40	parcela_unica	moveis_cozinha_area_de_servico_jantar_e_jardim	113
41	parcela_unica	construcao_ferramentas_jardim	113
42	parcela_unica	artes	111
43	parcela_unica	industria_comercio_e_negocios	105
44	parcela_unica	agro_industria_e_comercio	101
45	parcela_unica	fashion_calcados	95
46	parcela_unica	eletrodomesticos_2	90
47	parcela_unica	sinalizacao_e_seguranca	84
48	parcela_unica	construcao_ferramentas_seguranca	84
49	parcela_unica	artigos_de_natal	69
50	parcela_unica	fashion_underwear_e_moda_praia	68
51	parcela_unica	fashion_roupa_masculina	55
52	parcela_unica	construcao_ferramentas_ferramentas	47
53	parcela_unica	pcs	44
54	parcela_unica	dvds_blu_ray	42
55	parcela_unica	cine_foto	41
56	parcela_unica	moveis_quarto	40
57	parcela_unica	tablets_impresao_imagem	36
58	parcela_unica	livros_importados	33
59	parcela_unica	flores	26
60	parcela_unica	portateis_casa_forno_e_cafe	25
61	parcela_unica	artigos_de_festas	25
62	parcela_unica	musica	19
63	parcela_unica	casa_conforto_2	18
64	parcela_unica	artes_e_artesanato	18
65	parcela_unica	fashion_roupa_feminina	17
66	parcela_unica	fraldas_higiene	15

Figura 75 – Pedidos Pagos em Parcela Única - Parte 2.
Fonte: autoria própria

	payment_type text	count bigint	pagamento numeric
1	voucher	3866	375539.32
2	not_defined	3	0.00
3	debit_card	1528	215129.02
4	credit_card	76505	12447417.87
5	boleto	19784	2844306.40

Figura 76 – Número de Pedidos Bem-Sucedidos e Valor Total por Método de Pagamento. Fonte: autoria própria

	estado text	metodo_pagamento text	total_pedidos bigint	receita_total numeric
1	AC	credit_card	61	14625.41
2	AC	boleto	16	3908.46
3	AC	voucher	4	797.13
4	AC	debit_card	2	349.62
5	AL	credit_card	338	80287.53
6	AL	boleto	68	13742.62
7	AL	voucher	10	1705.79
8	AL	debit_card	5	1226.12
9	AM	credit_card	124	23151.97
10	AM	boleto	21	4452.12
11	AM	voucher	4	210.82
12	AM	debit_card	2	152.02
13	AP	credit_card	47	12199.02
14	AP	boleto	20	3826.36
15	AP	voucher	3	237.42
16	BA	credit_card	2643	498907.13
17	BA	boleto	614	95019.70
18	BA	voucher	176	15665.56
19	BA	debit_card	53	7053.43
20	CE	credit_card	1083	229041.53
21	CE	boleto	205	39728.69
22	CE	voucher	59	7620.62
23	CE	debit_card	20	3073.19
24	DF	credit_card	1695	291962.63
25	DF	boleto	396	52916.91
26	DF	voucher	79	8600.13
27	DF	debit_card	17	1661.41
28	ES	credit_card	1568	249150.28
29	ES	boleto	403	65566.33
30	ES	voucher	82	6651.12
31	ES	debit_card	27	4599.82
32	GO	credit_card	1514	269630.79
33	GO	boleto	449	68947.89

Figura 77 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - Parte 1. Fonte: autoria própria

34	GO	voucher	76	7982.69
35	GO	debit_card	22	3530.94
36	MA	credit_card	534	109067.63
37	MA	boleto	203	40689.92
38	MA	voucher	19	2356.11
39	MA	debit_card	4	409.36
40	MG	credit_card	9030	1471975.31
41	MG	boleto	2304	340518.70
42	MG	voucher	438	37533.85
43	MG	debit_card	139	22229.40
44	MS	credit_card	516	106565.45
45	MS	boleto	175	27596.79
46	MS	voucher	24	1878.73
47	MS	debit_card	11	1493.87
48	MT	credit_card	657	144971.86
49	MT	boleto	238	38875.67
50	MT	voucher	25	2342.78
51	MT	debit_card	6	838.98
52	PA	credit_card	724	166177.62
53	PA	boleto	215	44438.51
54	PA	voucher	41	4281.94
55	PA	debit_card	16	3397.78
56	PB	credit_card	421	107044.51
57	PB	boleto	93	26424.87
58	PB	debit_card	13	5426.04
59	PB	voucher	23	2650.30
60	PE	credit_card	1326	267609.24
61	PE	boleto	277	47887.11
62	PE	voucher	73	6477.01
63	PE	debit_card	19	2877.08
64	PI	credit_card	383	87538.00
65	PI	boleto	94	17424.15
66	PI	voucher	27	2329.86

Figura 78 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - Parte 2. Fonte: autoria própria

67	PI	debit_card	11	1231.96
68	PR	credit_card	3776	626606.74
69	PR	boleto	1118	155631.79
70	PR	voucher	199	18375.61
71	PR	debit_card	74	10542.24
72	RJ	credit_card	10253	1730340.68
73	RJ	boleto	2163	328621.78
74	RJ	voucher	547	52994.22
75	RJ	debit_card	185	32423.01
76	RN	credit_card	390	81441.78
77	RN	boleto	80	16470.25
78	RN	voucher	21	3308.37
79	RN	debit_card	9	1497.73
80	RO	credit_card	184	43316.56
81	RO	boleto	64	14810.29
82	RO	voucher	7	1988.39
83	RO	debit_card	3	750.96
84	RR	credit_card	33	7323.49
85	RR	boleto	13	2741.13
86	RS	credit_card	3967	662551.39
87	RS	boleto	1359	192327.91
88	RS	voucher	179	21563.66
89	RS	debit_card	76	14455.58
90	SC	credit_card	2705	481784.37
91	SC	boleto	840	114313.98
92	SC	voucher	127	19391.52
93	SC	debit_card	46	7596.56
94	SE	credit_card	263	56291.60
95	SE	boleto	75	14587.62
96	SE	voucher	15	2749.50
97	SE	debit_card	5	1617.53
98	SP	credit_card	32074	4676662.34
99	SP	boleto	8205	1084604.44

Figura 79 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - Parte 3. Fonte: autoria própria

100	SP	voucher	1596	148301.67
101	SP	debit_card	759	88658.51
102	SP	not_defined	3	0.00
103	TO	credit_card	196	45859.33
104	TO	boleto	76	13287.28
105	TO	voucher	12	1442.07
106	TO	debit_card	4	896.65

Figura 80 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - Parte 4. Fonte: autoria própria

	categoria text	receita_total numeric	tempo_medio_entrega_dias numeric
1	beleza_saude	1233211.61	11.98052502919498139776
2	relogios_presentes	1165898.98	12.66318005046193538612
3	cama_mesa_banho	1023434.76	12.80646297860217562603
4	esporte_lazer	954695.05	12.15205680809798055694
5	informatica_acessorios	888613.62	13.22530387433914353980

Figura 81 – Top 5 Categorias por Receita e Tempo Médio de Entrega. Fonte: autoria própria

A.2 Resultados das Consultas no MongoDB

Para documentar os resultados das dez consultas analíticas executadas no MongoDB, foram elaboradas as Figuras 79 a 89, nas quais se encontram os valores retornados por cada consulta. Na Figura 79, referente à consulta de “Total de Vendas por Estado do cliente” (Veja Seção 4.4.1), apresenta o valor total de vendas para cada estado, ordenado

de forma decrescente. Na Figura 82, referente à consulta de “Top 10 Produtos Mais Vendidos” (Veja Seção 4.4.2), apresenta a quantidade de vendas por categoria de produto, ordenado de forma decrescente. Na Figura 84, referente à consulta de “Receita Total por Categoria de Produto” (Veja Seção 4.4.3), apresenta o valor total de vendas para cada categoria de produto, ordenado de forma decrescente. Na Figura 87, referente à consulta de “Top 10 Vendedores por Número de Itens Vendidos” (Veja Seção 4.4.4), apresenta os `id` dos dez vendedores que mais venderam itens junto com seu estado de origem, cidade e a quantidade de itens vendidos, ordenado de forma decrescente. Na Figura 89, referente à consulta de “Top 5 Categorias mais Vendidas por Número de Pedidos” (Veja Seção 4.4.5), apresenta as Top 5 categorias que mais foram vendidas, ordenado de forma decrescente.

```

< {
  total_vendas: 5202955.05,
  state: 'SP'
}
{
  total_vendas: 1824092.67,
  state: 'RJ'
}
{
  total_vendas: 1585308.03,
  state: 'MG'
}
{
  total_vendas: 750304.02,
  state: 'RS'
}
{
  total_vendas: 683083.76,
  state: 'PR'
}
{
  total_vendas: 520553.34,
  state: 'SC'
}
{
  total_vendas: 511349.99,
  state: 'BA'
}
{
  total_vendas: 302603.94,
  state: 'DF'
}
{
  total_vendas: 294591.95,
  state: 'GO'
}

```

Figura 82 – Total de Vendas por Estado do Cliente - parte 1. Fonte: autoria própria

```

{
  total_vendas: 275037.31,
  state: 'ES'
}
{
  total_vendas: 262788.03,
  state: 'PE'
}
{
  total_vendas: 227254.71,
  state: 'CE'
}
{
  total_vendas: 178947.81,
  state: 'PA'
}
{
  total_vendas: 156453.53,
  state: 'MT'
}
{
  total_vendas: 119648.22,
  state: 'MA'
}
{
  total_vendas: 116812.64,
  state: 'MS'
}
{
  total_vendas: 115268.08,
  state: 'PB'
}
{
  total_vendas: 86914.08,
  state: 'PI'
}

```

Figura 83 – Total de Vendas por Estado do Cliente - parte 2. Fonte: autoria própria

```

{
  total_vendas: 83034.98,
  state: 'RN'
}
{
  total_vendas: 80314.81,
  state: 'AL'
}

```

Figura 84 – Total de Vendas por Estado do Cliente - parte 3. Fonte: autoria própria

```

< {
  quantidade_vendas: 11115,
  categoria_pt: 'cama_mesa_banho'
}
{
  quantidade_vendas: 9670,
  categoria_pt: 'beleza_saude'
}
{
  quantidade_vendas: 8641,
  categoria_pt: 'esporte_lazer'
}
{
  quantidade_vendas: 8334,
  categoria_pt: 'moveis_decoracao'
}
{
  quantidade_vendas: 7827,
  categoria_pt: 'informatica_acessorios'
}
{
  quantidade_vendas: 6964,
  categoria_pt: 'utilidades_domesticas'
}
{
  quantidade_vendas: 5991,
  categoria_pt: 'relogios_presentes'
}
{
  quantidade_vendas: 4545,
  categoria_pt: 'telefonica'
}
{
  quantidade_vendas: 4347,
  categoria_pt: 'ferramentas_jardim'
}

```

Figura 85 – Produtos Mais Vendidos (Top 10) - parte 1.
Fonte: autoria própria

```

{
  quantidade_vendas: 4235,
  categoria_pt: 'automotivo'
}

```

Figura 86 – Produtos Mais Vendidos (Top 10) - parte 2.
Fonte: autoria própria

```

< {
  receita_total: 1258681.34,
  categoria_pt: 'beleza_saude'
}
{
  receita_total: 1205005.68,
  categoria_pt: 'relogios_presentes'
}
{
  receita_total: 1036988.68,
  categoria_pt: 'cama_mesa_banho'
}
{
  receita_total: 988048.97,
  categoria_pt: 'esporte_lazer'
}
{
  receita_total: 911954.32,
  categoria_pt: 'informatica_acessorios'
}
{
  receita_total: 729762.49,
  categoria_pt: 'moveis_decoracao'
}
{
  receita_total: 635290.85,
  categoria_pt: 'cool_stuff'
}
{
  receita_total: 632248.66,
  categoria_pt: 'utilidades_domesticas'
}
{
  receita_total: 592720.11,
  categoria_pt: 'automotivo'
}
}

```

Figura 87 – Receita Total por Categoria de Produto - Parte 1. Fonte: autoria própria

```

{
  receita_total: 485256.46,
  categoria_pt: 'ferramentas_jardim'
}
{
  receita_total: 483946.6,
  categoria_pt: 'brinquedos'
}
{
  receita_total: 411764.89,
  categoria_pt: 'bebes'
}
{
  receita_total: 399124.87,
  categoria_pt: 'perfumaria'
}
{
  receita_total: 323667.52999999997,
  categoria_pt: 'telefonica'
}
{
  receita_total: 273960.7,
  categoria_pt: 'moveis_escritorio'
}
{
  receita_total: 230943.23,
  categoria_pt: 'papelaria'
}
{
  receita_total: 222963.13,
  categoria_pt: 'pcs'
}
{
  receita_total: 214315.41,
  categoria_pt: 'pet_shop'
}
}

```

Figura 88 – Receita Total por Categoria de Produto - Parte 2. Fonte: autoria própria

```

{
  receita_total: 191498.88,
  categoria_pt: 'instrumentos_musicais'
}
{
  receita_total: 190648.58,
  categoria_pt: 'eletroportateis'
}
}

```

Figura 89 – Receita Total por Categoria de Produto - Parte 3. Fonte: autoria própria

```

< {
  itens_vendidos: 2033,
  seller_id: '6560211a19b47992c3666cc44a7e94c8',
  seller_city: 'sao paulo',
  seller_state: 'SP'
}
{
  itens_vendidos: 1987,
  seller_id: '4a3ca9315b744ce9f8e9374361493884',
  seller_city: 'ibitinga',
  seller_state: 'SP'
}
{
  itens_vendidos: 1931,
  seller_id: '1f50f920176fa81dab994f9023523100',
  seller_city: 'sao jose do rio preto',
  seller_state: 'SP'
}
{
  itens_vendidos: 1775,
  seller_id: 'cc419e0650a3c5ba77189a1882b7556a',
  seller_city: 'santo andre',
  seller_state: 'SP'
}
{
  itens_vendidos: 1551,
  seller_id: 'da8622b14eb17ae2831f4ac5b9dab84a',
  seller_city: 'piracicaba',
  seller_state: 'SP'
}
{
  itens_vendidos: 1499,
  seller_id: '955fee9216a65b617aa5c0531780ce60',
  seller_city: 'sao paulo',
  seller_state: 'SP'
}
}

```

Figura 90 – Top 10 Vendedores por Número de Itens Vendidos - parte 1. Fonte: autoria própria

```

{
  itens_vendidos: 1428,
  seller_id: '1025f0e2d44d7941d6cf58b6550e0bfa',
  seller_city: 'sao paulo',
  seller_state: 'SP'
}
{
  itens_vendidos: 1364,
  seller_id: '7c67e1448b00f6e969d365cea6b010ab',
  seller_city: 'itaquaquecetuba',
  seller_state: 'SP'
}
{
  itens_vendidos: 1203,
  seller_id: 'ea8482cd71df3c1969d7b9473ff13abc',
  seller_city: 'sao paulo',
  seller_state: 'SP'
}
{
  itens_vendidos: 1171,
  seller_id: '7a67c85e85bb2ce8502c35f2203ad736',
  seller_city: 'sao paulo',
  seller_state: 'SP'
}
}

```

Figura 91 – Top 10 Vendedores por Número de Itens Vendidos - parte 2. Fonte: autoria própria

```

< {
  total_pedidos: 9417,
  categoria: 'cama_mesa_banho'
}
{
  total_pedidos: 8836,
  categoria: 'beleza_saude'
}
{
  total_pedidos: 7720,
  categoria: 'esporte_lazer'
}
{
  total_pedidos: 6689,
  categoria: 'informatica_acessorios'
}
{
  total_pedidos: 6449,
  categoria: 'moveis_decoracao'
}
}

```

Figura 92 – Top 5 Categorias mais Vendidas por Número de Pedidos. Fonte: autoria própria

Na sequência, a Figura 90, referente à consulta de “Análise de Estados com Maior Número de Pedidos em Parcelas Acima de 12 Vezes” (Veja Seção 4.4.6), apresenta a quantidade de pedidos feitos por estado com mais de doze parcelas, ordenado de forma decrescente. Na Figura 93, referente à consulta de “Pedidos Pagos em Parcela Única” (Veja Seção 4.4.7), apresenta a quantidade total de pedidos para cada categoria, ordenado de forma decrescente. Na Figura 96, referente à consulta de “Número de Pedidos Bem-Sucedidos e Valor Total por Método de Pagamento” (Veja Seção 4.4.8), apresenta a quantidade de pedidos efetivados por método de pagamento e valor total. Na Figura 98, referente à consulta de “Receita e Quantidade de Pedidos por Estado e Método de Pagamento” (Veja Seção 4.4.9), apresenta o valor total e a quantidade de pedidos feitos por estado para cada método de pagamento e ordenado de forma decrescente. Na Figura 118, referente à consulta de “Top 5 Categorias por Receita e Tempo Médio de Entrega” (Veja Seção 4.4.10), apresenta as cinco melhores categorias que obtiveram mais lucro e a média de entrega.

```
< {
  num_orders: 62,
  state: 'SP'
}
{
  num_orders: 26,
  state: 'RJ'
}
{
  num_orders: 19,
  state: 'MG'
}
{
  num_orders: 16,
  state: 'RS'
}
{
  num_orders: 13,
  state: 'PR'
}
{
  num_orders: 13,
  state: 'BA'
}
{
  num_orders: 7,
  state: 'PE'
}
{
  num_orders: 5,
  state: 'CE'
}
{
  num_orders: 4,
  state: 'ES'
}
```

Figura 93 – Regiões onde Residem os Usuários com o Maior Número de Parcelas - parte 1. Fonte:

```
{
  num_orders: 4,
  state: 'PB'
}
{
  num_orders: 3,
  state: 'PA'
}
{
  num_orders: 3,
  state: 'GO'
}
{
  num_orders: 3,
  state: 'SC'
}
{
  num_orders: 2,
  state: 'DF'
}
{
  num_orders: 1,
  state: 'MA'
}
{
  num_orders: 1,
  state: 'AL'
}
{
  num_orders: 1,
  state: 'SE'
}
{
  num_orders: 1,
  state: 'RO'
}
```

Figura 94 – Regiões onde Residem os Usuários com o Maior Número de Parcelas - parte 2.

```
{
  num_orders: 1,
  state: 'PI'
}
```

Figura 95 – Regiões onde Residem os Usuários com o Maior Número de Parcelas - parte 3. Fonte: autoria própria

centering

```
< {
  categoria: 'esporte_lazer',
  total_pedidos: 4299
}
{
  categoria: 'informatica_acessorios',
  total_pedidos: 4177
}
{
  categoria: 'beleza_saude',
  total_pedidos: 3880
}
{
  categoria: 'cama_mesa_banho',
  total_pedidos: 3535
}
{
  categoria: 'moveis_decoracao',
  total_pedidos: 3152
}
{
  categoria: 'utilidades_domesticas',
  total_pedidos: 2732
}
{
  categoria: 'telefonica',
  total_pedidos: 2552
}
{
  categoria: 'eletronicos',
  total_pedidos: 2052
}
{
  categoria: 'automotivo',
  total_pedidos: 2041
}
}
```

Figura 96 – Pedidos Pagos em Parcela Única - Parte 1. Fonte: autoria própria

```
{
  categoria: 'brinquedos',
  total_pedidos: 1902
}
{
  categoria: 'relogios_presentes',
  total_pedidos: 1890
}
{
  categoria: 'ferramentas_jardim',
  total_pedidos: 1814
}
{
  categoria: 'cool_stuff',
  total_pedidos: 1451
}
{
  categoria: 'bebes',
  total_pedidos: 1386
}
{
  categoria: 'perfumaria',
  total_pedidos: 1238
}
{
  categoria: 'papelaria',
  total_pedidos: 1153
}
{
  categoria: 'fashion_bolsas_e_acessorios',
  total_pedidos: 999
}
{
  categoria: 'pet_shop',
  total_pedidos: 852
}
}
```

Figura 97 – Pedidos Pagos em Parcela Única - Parte 2. Fonte: autoria própria

```
{
  categoria: null,
  total_pedidos: 741
}
{
  categoria: 'eletrodomesticos',
  total_pedidos: 585
}
```

Figura 98 – Pedidos Pagos em Parcela Única - Parte 3. Fonte: autoria própria

```
{
  pagamento_total: 375539.32,
  metodo: 'voucher',
  total_pedidos: 3866
}
{
  pagamento_total: 0,
  metodo: 'not_defined',
  total_pedidos: 3
}
{
  pagamento_total: 215129.02,
  metodo: 'debit_card',
  total_pedidos: 1528
}
{
  pagamento_total: 12447417.87,
  metodo: 'credit_card',
  total_pedidos: 76505
}
{
  pagamento_total: 2844306.4,
  metodo: 'boleto',
  total_pedidos: 19784
}
```

Figura 99 – Número de Pedidos Bem-Sucedidos e Valor Total por Método de Pagamento. Fonte: autoria própria

```
< {
  receita_total: 12400.18,
  estado: 'AC',
  metodo_pagamento: 'credit_card',
  total_orders: 61
}
{
  receita_total: 3236.98,
  estado: 'AC',
  metodo_pagamento: 'boleto',
  total_orders: 16
}
{
  receita_total: 1173.29,
  estado: 'AC',
  metodo_pagamento: 'voucher',
  total_orders: 4
}
{
  receita_total: 248.99,
  estado: 'AC',
  metodo_pagamento: 'debit_card',
  total_orders: 2
}
{
  receita_total: 68473.58,
  estado: 'AL',
  metodo_pagamento: 'credit_card',
  total_orders: 338
}
```

Figura 100 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento. Fonte: autoria própria

```

{
  receita_total: 11318.35,
  estado: 'AL',
  metodo_pagamento: 'boleto',
  total_orders: 68
}
{
  receita_total: 2301.55,
  estado: 'AL',
  metodo_pagamento: 'voucher',
  total_orders: 10
}
{
  receita_total: 1016.88,
  estado: 'AL',
  metodo_pagamento: 'debit_card',
  total_orders: 5
}
{
  receita_total: 18698.71,
  estado: 'AM',
  metodo_pagamento: 'credit_card',
  total_orders: 124
}
{
  receita_total: 3524.15,
  estado: 'AM',
  metodo_pagamento: 'boleto',
  total_orders: 21
}
{
  receita_total: 480.51,
  estado: 'AM',
  metodo_pagamento: 'voucher',
  total_orders: 4
}

```

Figura 101 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - parte 1. Fonte: autoria própria

```

{
  receita_total: 105.99,
  estado: 'AM',
  metodo_pagamento: 'debit_card',
  total_orders: 2
}
{
  receita_total: 10515.7,
  estado: 'AP',
  metodo_pagamento: 'credit_card',
  total_orders: 47
}
{
  receita_total: 2908.61,
  estado: 'AP',
  metodo_pagamento: 'boleto',
  total_orders: 20
}
{
  receita_total: 229.99,
  estado: 'AP',
  metodo_pagamento: 'voucher',
  total_orders: 3
}
{
  receita_total: 429987.62,
  estado: 'BA',
  metodo_pagamento: 'credit_card',
  total_orders: 2643
}

```

Figura 102 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - parte 2. Fonte: autoria própria

```

{
  receita_total: 75103.44,
  estado: 'BA',
  metodo_pagamento: 'boleto',
  total_orders: 614
}
{
  receita_total: 30685.5,
  estado: 'BA',
  metodo_pagamento: 'voucher',
  total_orders: 176
}
{
  receita_total: 5634.64,
  estado: 'BA',
  metodo_pagamento: 'debit_card',
  total_orders: 53
}
{
  receita_total: 192708.36,
  estado: 'CE',
  metodo_pagamento: 'credit_card',
  total_orders: 1083
}

```

Figura 103 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - parte 3. Fonte: autoria própria

```

< {
  receita_total: 32511.22,
  estado: 'CE',
  metodo_pagamento: 'boleto',
  total_orders: 205
}
{
  receita_total: 11835.09,
  estado: 'CE',
  metodo_pagamento: 'voucher',
  total_orders: 59
}
{
  receita_total: 2363.57,
  estado: 'CE',
  metodo_pagamento: 'debit_card',
  total_orders: 20
}
{
  receita_total: 258669.73,
  estado: 'DF',
  metodo_pagamento: 'credit_card',
  total_orders: 1695
}
{
  receita_total: 43398.01,
  estado: 'DF',
  metodo_pagamento: 'boleto',
  total_orders: 396
}

```

Figura 104 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - parte 4. Fonte: autoria própria

```

{
  receita_total: 9694,
  estado: 'DF',
  metodo_pagamento: 'voucher',
  total_orders: 79
}
{
  receita_total: 1306.32,
  estado: 'DF',
  metodo_pagamento: 'debit_card',
  total_orders: 17
}
{
  receita_total: 215345.3,
  estado: 'ES',
  metodo_pagamento: 'credit_card',
  total_orders: 1568
}
{
  receita_total: 55543.34,
  estado: 'ES',
  metodo_pagamento: 'boleto',
  total_orders: 403
}
{
  receita_total: 9013.31,
  estado: 'ES',
  metodo_pagamento: 'voucher',
  total_orders: 82
}
{
  receita_total: 3995.98,
  estado: 'ES',
  metodo_pagamento: 'debit_card',
  total_orders: 27
}

```

Figura 105 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - parte 5. Fonte: autoria própria

```

{
  receita_total: 233526.87,
  estado: 'GO',
  metodo_pagamento: 'credit_card',
  total_orders: 1514
}
{
  receita_total: 57334.44,
  estado: 'GO',
  metodo_pagamento: 'boleto',
  total_orders: 449
}
{
  receita_total: 15984.55,
  estado: 'GO',
  metodo_pagamento: 'voucher',
  total_orders: 76
}
{
  receita_total: 2988.36,
  estado: 'GO',
  metodo_pagamento: 'debit_card',
  total_orders: 22
}
{
  receita_total: 86752.7,
  estado: 'MA',
  metodo_pagamento: 'credit_card',
  total_orders: 534
}

```

Figura 106 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - parte 6. Fonte: autoria própria

```

{
  receita_total: 31972.96,
  estado: 'MA',
  metodo_pagamento: 'boleto',
  total_orders: 203
}
{
  receita_total: 3912.16,
  estado: 'MA',
  metodo_pagamento: 'voucher',
  total_orders: 19
}
{
  receita_total: 243.97,
  estado: 'MA',
  metodo_pagamento: 'debit_card',
  total_orders: 4
}
{
  receita_total: 1280183.3,
  estado: 'MG',
  metodo_pagamento: 'credit_card',
  total_orders: 9030
}

```

Figura 107 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - parte 7.
Fonte: autoria própria

```

< {
  receita_total: 284540.33,
  estado: 'MG',
  metodo_pagamento: 'boleto',
  total_orders: 2304
}
{
  receita_total: 55792.57,
  estado: 'MG',
  metodo_pagamento: 'voucher',
  total_orders: 438
}
{
  receita_total: 19120.63,
  estado: 'MG',
  metodo_pagamento: 'debit_card',
  total_orders: 139
}
{
  receita_total: 93528.34,
  estado: 'MS',
  metodo_pagamento: 'credit_card',
  total_orders: 516
}
{
  receita_total: 22859.83,
  estado: 'MS',
  metodo_pagamento: 'boleto',
  total_orders: 175
}
{
  receita_total: 2270.86,
  estado: 'MS',
  metodo_pagamento: 'voucher',
  total_orders: 24
}

```

Figura 108 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - parte 8.
Fonte: autoria própria

```

{
  receita_total: 1164.3799999999999,
  estado: 'MS',
  metodo_pagamento: 'debit_card',
  total_orders: 11
}
{
  receita_total: 125204.46,
  estado: 'MT',
  metodo_pagamento: 'credit_card',
  total_orders: 657
}
{
  receita_total: 31604.46,
  estado: 'MT',
  metodo_pagamento: 'boleto',
  total_orders: 238
}
{
  receita_total: 13368.94,
  estado: 'MT',
  metodo_pagamento: 'voucher',
  total_orders: 25
}
{
  receita_total: 644.1800000000001,
  estado: 'MT',
  metodo_pagamento: 'debit_card',
  total_orders: 6
}
{
  receita_total: 140540.03,
  estado: 'PA',
  metodo_pagamento: 'credit_card',
  total_orders: 724
}

```

Figura 109 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - parte 9. Fonte: autoria própria

```

{
  receita_total: 35689.76,
  estado: 'PA',
  metodo_pagamento: 'boleto',
  total_orders: 215
}
{
  receita_total: 5437.88,
  estado: 'PA',
  metodo_pagamento: 'voucher',
  total_orders: 41
}
{
  receita_total: 2740.21,
  estado: 'PA',
  metodo_pagamento: 'debit_card',
  total_orders: 16
}
{
  receita_total: 94148.83,
  estado: 'PB',
  metodo_pagamento: 'credit_card',
  total_orders: 421
}
{
  receita_total: 19953.87,
  estado: 'PB',
  metodo_pagamento: 'boleto',
  total_orders: 93
}
{
  receita_total: 4853.36,
  estado: 'PB',
  metodo_pagamento: 'voucher',
  total_orders: 23
}

```

Figura 110 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - parte 10. Fonte: autoria própria

```
{
  receita_total: 4770.28,
  estado: 'PB',
  metodo_pagamento: 'debit_card',
  total_orders: 13
}
{
  receita_total: 222864.73,
  estado: 'PE',
  metodo_pagamento: 'credit_card',
  total_orders: 1326
}
```

Figura 111 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - parte 11. Fonte: autoria própria

```
< {
  receita_total: 37393.44,
  estado: 'PE',
  metodo_pagamento: 'boleto',
  total_orders: 277
}
{
  receita_total: 8796.6,
  estado: 'PE',
  metodo_pagamento: 'voucher',
  total_orders: 73
}
{
  receita_total: 2203.59,
  estado: 'PE',
  metodo_pagamento: 'debit_card',
  total_orders: 19
}
{
  receita_total: 74666.16,
  estado: 'PI',
  metodo_pagamento: 'credit_card',
  total_orders: 383
}
{
  receita_total: 13259.19,
  estado: 'PI',
  metodo_pagamento: 'boleto',
  total_orders: 94
}
{
  receita_total: 3328.75,
  estado: 'PI',
  metodo_pagamento: 'voucher',
  total_orders: 27
}
```

Figura 112 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - parte 12. Fonte: autoria própria

```

{
  receita_total: 913.6,
  estado: 'PI',
  metodo_pagamento: 'debit_card',
  total_orders: 11
}
{
  receita_total: 541492.99,
  estado: 'PR',
  metodo_pagamento: 'credit_card',
  total_orders: 3776
}
{
  receita_total: 128176.18,
  estado: 'PR',
  metodo_pagamento: 'boleto',
  total_orders: 1118
}
{
  receita_total: 27071.19,
  estado: 'PR',
  metodo_pagamento: 'voucher',
  total_orders: 199
}
{
  receita_total: 9116.11,
  estado: 'PR',
  metodo_pagamento: 'debit_card',
  total_orders: 74
}

```

Figura 113 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - parte 13.
Fonte: autoria própria

```

{
  receita_total: 1511935.17,
  estado: 'RJ',
  metodo_pagamento: 'credit_card',
  total_orders: 10253
}
{
  receita_total: 273654.84,
  estado: 'RJ',
  metodo_pagamento: 'boleto',
  total_orders: 2163
}
{
  receita_total: 99649.82,
  estado: 'RJ',
  metodo_pagamento: 'voucher',
  total_orders: 547
}
{
  receita_total: 28324.33,
  estado: 'RJ',
  metodo_pagamento: 'debit_card',
  total_orders: 185
}
{
  receita_total: 67749.6,
  estado: 'RN',
  metodo_pagamento: 'credit_card',
  total_orders: 390
}
{
  receita_total: 13480.22,
  estado: 'RN',
  metodo_pagamento: 'boleto',
  total_orders: 80
}

```

Figura 114 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - parte 14.
Fonte: autoria própria

```
{
  receita_total: 12098.74,
  estado: 'RN',
  metodo_pagamento: 'voucher',
  total_orders: 21
}
{
  receita_total: 1225.99,
  estado: 'RN',
  metodo_pagamento: 'debit_card',
  total_orders: 9
}
{
  receita_total: 34730.96,
  estado: 'RO',
  metodo_pagamento: 'credit_card',
  total_orders: 184
}
```

Figura 115 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - parte 15. Fonte: autoria própria

```
< {
  receita_total: 10820,
  estado: 'RO',
  metodo_pagamento: 'boleto',
  total_orders: 64
}
{
  receita_total: 835.17,
  estado: 'RO',
  metodo_pagamento: 'voucher',
  total_orders: 7
}
{
  receita_total: 577.9,
  estado: 'RO',
  metodo_pagamento: 'debit_card',
  total_orders: 3
}
{
  receita_total: 5734.9,
  estado: 'RR',
  metodo_pagamento: 'credit_card',
  total_orders: 33
}
{
  receita_total: 2094.5299999999997,
  estado: 'RR',
  metodo_pagamento: 'boleto',
  total_orders: 13
}
```

Figura 116 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - parte 16. Fonte: autoria própria

```

{
  receita_total: 579312.01,
  estado: 'RS',
  metodo_pagamento: 'credit_card',
  total_orders: 3967
}
{
  receita_total: 156757.24,
  estado: 'RS',
  metodo_pagamento: 'boleto',
  total_orders: 1359
}
{
  receita_total: 38996.26,
  estado: 'RS',
  metodo_pagamento: 'voucher',
  total_orders: 179
}
{
  receita_total: 12705.02,
  estado: 'RS',
  metodo_pagamento: 'debit_card',
  total_orders: 76
}
{
  receita_total: 418921.25,
  estado: 'SC',
  metodo_pagamento: 'credit_card',
  total_orders: 2705
}
{
  receita_total: 93532.89,
  estado: 'SC',
  metodo_pagamento: 'boleto',
  total_orders: 840
}

```

Figura 117 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - parte 17. Fonte: autoria própria

```

{
  receita_total: 19520.62,
  estado: 'SC',
  metodo_pagamento: 'voucher',
  total_orders: 127
}
{
  receita_total: 6240.69,
  estado: 'SC',
  metodo_pagamento: 'debit_card',
  total_orders: 46
}
{
  receita_total: 46713.72,
  estado: 'SE',
  metodo_pagamento: 'credit_card',
  total_orders: 263
}
{
  receita_total: 10064.87,
  estado: 'SE',
  metodo_pagamento: 'boleto',
  total_orders: 75
}
{
  receita_total: 2768.21,
  estado: 'SE',
  metodo_pagamento: 'voucher',
  total_orders: 15
}

```

Figura 118 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - parte 18. Fonte: autoria própria

```
{
  receita_total: 1407.8,
  estado: 'SE',
  metodo_pagamento: 'debit_card',
  total_orders: 5
}
{
  receita_total: 4171072.28,
  estado: 'SP',
  metodo_pagamento: 'credit_card',
  total_orders: 32074
}
{
  receita_total: 930184.97,
  estado: 'SP',
  metodo_pagamento: 'boleto',
  total_orders: 8205
}
{
  receita_total: 272854.86,
  estado: 'SP',
  metodo_pagamento: 'voucher',
  total_orders: 1596
}
```

Figura 119 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - parte 19. Fonte: autoria própria

```
< {
  receita_total: 74054.64,
  estado: 'SP',
  metodo_pagamento: 'debit_card',
  total_orders: 759
}
{
  receita_total: 0,
  estado: 'SP',
  metodo_pagamento: 'not_defined',
  total_orders: 3
}
{
  receita_total: 38479.82,
  estado: 'TO',
  metodo_pagamento: 'credit_card',
  total_orders: 196
}
{
  receita_total: 10607.54,
  estado: 'TO',
  metodo_pagamento: 'boleto',
  total_orders: 76
}
{
  receita_total: 6519.86,
  estado: 'TO',
  metodo_pagamento: 'voucher',
  total_orders: 12
}
{
  receita_total: 644.69,
  estado: 'TO',
  metodo_pagamento: 'debit_card',
  total_orders: 4
}
```

Figura 120 – Receita e Quantidade de Pedidos por Estado e Método de Pagamento - parte 20. Fonte: autoria própria

```
< {  
  receita_total: 1233211.61,  
  categoria: 'beleza_saude',  
  tempo_medio_entrega: 11.98  
}  
{  
  receita_total: 1165898.98,  
  categoria: 'relogios_presentes',  
  tempo_medio_entrega: 12.66  
}  
{  
  receita_total: 1023434.76,  
  categoria: 'cama_mesa_banho',  
  tempo_medio_entrega: 12.81  
}  
{  
  receita_total: 954695.05,  
  categoria: 'esporte_lazer',  
  tempo_medio_entrega: 12.15  
}  
{  
  receita_total: 888613.62,  
  categoria: 'informatica_acessorios',  
  tempo_medio_entrega: 13.23  
}
```

Figura 121 – Top 5 Categorias por Receita e Tempo Médio de Entrega. Fonte: autoria própria