Final Proposal: ACID vs. BASE Properties Comparison Using Ecommerce DATASET

Introduction

This project evaluates the strengths and weaknesses of PostgreSQL (relational, ACID-compliant) and MongoDB (NoSQL, BASE-compliant) within the context of an e-commerce platform. Using the Brazilian E-commerce Dataset (Olist), we compared both databases across five critical metrics: **Performance, Scalability, Flexibility, Complexity, and Functionality**. Real-world scenarios such as **order processing, payment tracking, and review sentiment analysis** were implemented to assess their effectiveness.

Our aim is not to declare one database superior but to provide a nuanced understanding of their **trade-offs**, focusing on the dataset's characteristics and operational needs.

Dataset Overview and Relationships

Dataset Link: https://www.kaggle.com/datasets/olistbr/brazilian-ecommerce?resource=download

This project utilizes the **Brazilian E-commerce (Olist) dataset**, which consists of multiple linked tables representing real-world e-commerce operations. Key tables include:

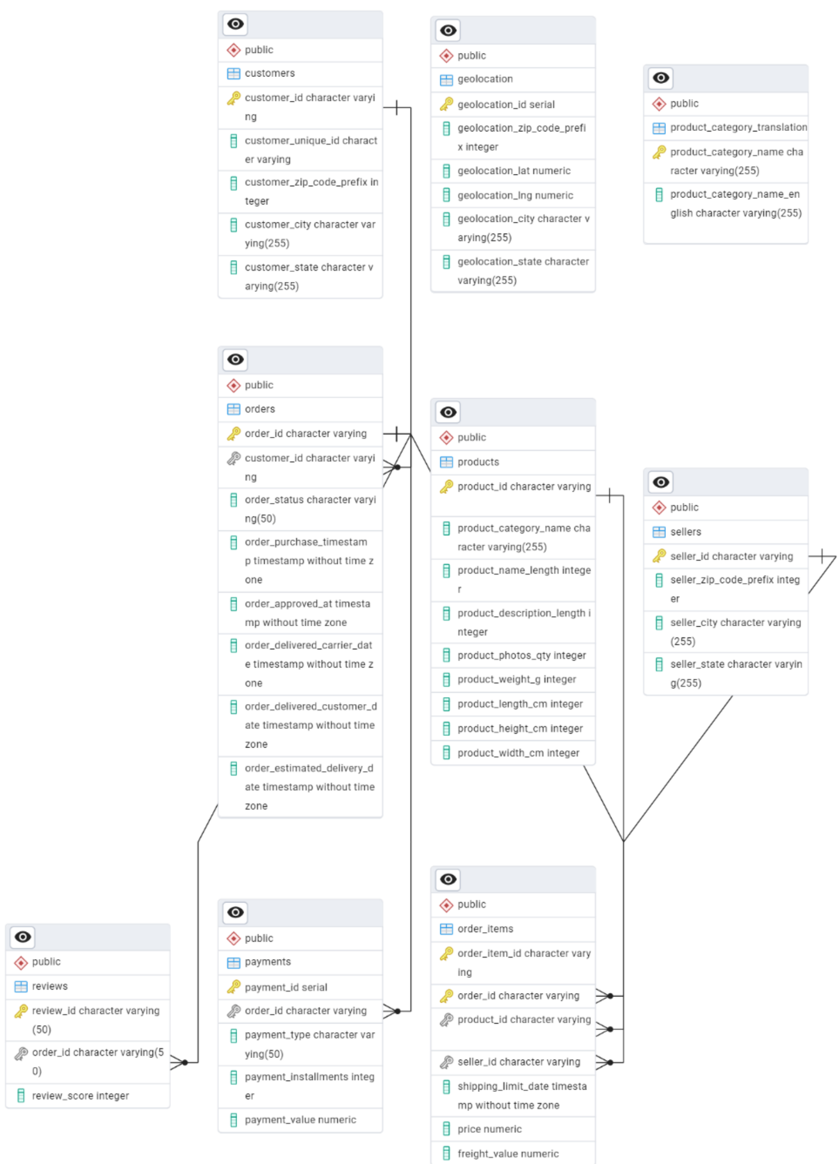
1. **Customers**: Contains customer demographics and location data.
2. **Orders**: Tracks order details such as timestamps and statuses.
3. **Order Items**: Maps each order to multiple products, including pricing and freight values.
4. **Products**: Holds product metadata such as dimensions and categories.
5. **Sellers**: Includes seller location and identifiers.
6. **Payments**: Details transaction types, values, and installment counts.
7. **Reviews**: Captures customer feedback with review scores and comments.
8. **Geolocation**: Maps ZIP codes to city and state information.
9. **Product Category Translation**: Translates product categories into human-readable labels.

Relationships Between Tables:

- The **Orders** table connects to **Customers** and **Order Items**, enabling analysis of purchase behavior.
- The **Order Items** table links products to sellers, reflecting the supply chain.
- **Reviews** are linked to **Orders**, facilitating sentiment analysis.
- **Geolocation** supports geographical insights by connecting with **Customers** and **Sellers**.

The relational structure in PostgreSQL and the schema-less, denormalized format in MongoDB were leveraged to test **ACID** and **BASE** properties under various scenarios.

ERD of Dataset



Why Denormalization Was Needed for MongoDB

Challenges with Normalization in MongoDB:

1. **Lack of Native Joins**: MongoDB does not natively support relational joins like SQL databases, making multi-collection queries inefficient.
2. **Read-Heavy Operations**: Joins between multiple collections increase query latency, particularly for large datasets.
3. **Distributed Environment**: Normalized designs complicate sharding and scaling in MongoDB.

Advantages:

- Faster read operations by eliminating cross-collection lookups.
- Simplified aggregation pipelines for performance-critical tasks.
- Optimized for MongoDB's **eventual consistency** and horizontal scaling.

Implemented Scenarios

1. **Order Processing**:
    - PostgreSQL: Identified peak ordering hours using advanced SQL functions like RANK and Common Table Expressions (CTEs).

```
WITH HourlyOrders AS (
  -- Extract the hour from the order purchase timestamp and count orders for each hour
  SELECT
    EXTRACT(HOUR FROM order_purchase_timestamp) AS order_hour,
    COUNT(*) AS order_count
  FROM orders
  GROUP BY order_hour
),
RankedOrders AS (
  -- Rank the hours based on the order count, in descending order
  SELECT
    order_hour,
    order_count,
    RANK() OVER (ORDER BY order_count DESC) AS rank
  FROM HourlyOrders
```

```
)
SELECT
   order_hour,
   order_count
FROM RankedOrders
WHERE rank = 1;  -- Retrieve the peak hour(s) based on the highest number of orders
```

Data Output | Messages | Notifications

	order_hour numeric	order_count bigint
1	16	6675

- MongoDB: Used aggregation pipelines ($group, $sort) to determine peak ordering times.

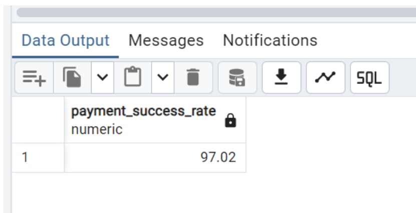
```
db.order_items_products.aggregate([
   { $group: { _id: "$product_category_name", totalOrders: { $sum: 1 } } },
   { $sort: { totalOrders: -1 } }
]);
```

```
>_MONGOSH
< {
    _id: 'cama_mesa_banho',
    totalOrders: 11115
  }
  {
    _id: 'beleza_saude',
    totalOrders: 9670
  }
  {
    _id: 'esporte_lazer',
    totalOrders: 8641
  }
  {
    _id: 'moveis_decoracao',
    totalOrders: 8334
  }
  {
    _id: 'informatica_acessorios',
    totalOrders: 7827
  }
  {
    _id: 'utilidades_domesticas',
    totalOrders: 6964
  }
  {
    _id: 'relogios_presentes',
    totalOrders: 5991
  }
  {
    _id: 'telefonia',
    totalOrders: 4545
  }
  {
    _id: 'ferramentas_jardim',
    totalOrders: 4347
  }
  {
```

2. **Payment Success Rates**:

- PostgreSQL: Calculated delivery and payment success percentages using conditional aggregates.

```
SELECT
  ROUND((COUNT(*) FILTER (WHERE order_status = 'delivered')::DECIMAL / COUNT(*) * 100), 2) AS
payment_success_rate
FROM orders
```



- MongoDB: Leveraged $cond in aggregation pipelines for payment success analysis.

```
db.orders_customers.aggregate([
  {
    $group: {
      _id: null,
      totalOrders: { $sum: 1 }, // Total number of orders
      deliveredOrders: { $sum: { $cond: [{ $eq: ["$order_status", "delivered"] }, 1, 0] } } // Count of delivered orders
    }
  },
  {
    $project: {
      _id: 0,
      paymentSuccessRate: {
        $multiply: [
          { $divide: ["$deliveredOrders", "$totalOrders"] },
          100
        ]
      }
    }
  }
```


```
]);
```

3. **Review Sentiment Analysis**:

- PostgreSQL: Used CASE statements to classify reviews as Positive, Neutral, or Negative.

```
SELECT
  p.product_id,
  p.product_category_name,
  r.order_id,
  r.review_score,
  CASE
    WHEN r.review_score = 3 THEN 'Neutral'
    WHEN r.review_score > 3 THEN 'Positive'
    ELSE 'Negative'
  END AS sentiment
FROM public.reviews r
JOIN public.orders o ON r.order_id = o.order_id
JOIN public.order_items oi ON o.order_id = oi.order_id
JOIN public.products p ON oi.product_id = p.product_id;
```

Data Output Messages Notifications

	product_id character varying	product_category_name character varying (255)	order_id character varying (50)	review_score integer	sentiment text
1	4244733e06e7ecb4970a6e2683c13e…	cool_stuff	00010242fe8c5a6d1ba2dd792cb16214	5	Positive
2	e5f2d52b802189ee658865ca93d83a…	pet_shop	00018f77f2f0320c557190d7a144bdd3	4	Positive
3	c777355d18b72b67abbeef9df44fd0fd	moveis_decoracao	000229ec398224ef6ca0657da4fc703e	5	Positive
4	7634da152a4610f1595efa32f14722fc	perfumaria	00024acbcdf0a6daa1e931b038114c75	4	Positive
5	ac6c3623068f30de03045865e4e100…	ferramentas_jardim	00042b26cf59d7ce69dfabb4e55b4fd9	5	Positive
6	ef92defde845ab8450f9d70c526ef70f	utilidades_domesticas	00048cc3ae777c65dbb7d2a0634bc1ea	4	Positive
7	8d4f2bb7e93e6710a28f34fa83ee7d28	telefonia	00054e8431b9d7675808bcb819fb4a32	4	Positive
8	557d850972a7d6f792fd18ae1400d9…	ferramentas_jardim	000576fe39319847cbb9d288c5617fa6	5	Positive
9	310ae3c140ff94b03219ad0adc3c778f	beleza_saude	0005a1a1728c9d785b8e2b08b90457…	1	Negative
10	4535b0e1091c278dfd193e5a1d63b3…	livros_tecnicos	0005f50442cb953dcd1d21e1fb923495	4	Positive
11	d63c1011f49d98b976c352955b1c4b…	beleza_saude	00061f2a7bc09da83e415a52dc8a4af1	5	Positive
12	f177554ea93259a5b282f24e33f65ab6	fashion_bolsas_e_acessorios	00063b381e2406b52ad429470734eb…	5	Positive
13	99a4788cb24856965c36a24e339b6…	cama_mesa_banho	0006ec9db01a64e59a68b2c340bf65a7	5	Positive
14	368c6c730842d78016ad823897a37…	ferramentas_jardim	0008288aa423d2a3f00fcb17cd7d8719	5	Positive
15	368c6c730842d78016ad823897a37…	ferramentas_jardim	0008288aa423d2a3f00fcb17cd7d8719	5	Positive
16	8cab8abac59158715e0d70a36c8074…	esporte_lazer	0009792311464db532ff765bf7b182ae	5	Positive

Total rows: 1000 of 111461 Query complete 00:00:00.615 Ln 1, Col 1

- MongoDB: Applied $project and $cond for dynamic review classification.

```
db.reviews_orders.aggregate([
  {
    $project: {
      review_id: 1,
      order_id: 1,
      review_score: 1,
      sentiment: {
        $cond: [
          { $eq: ["$review_score", 3] },
          "Neutral",
          {
            $cond: [
              { $gt: ["$review_score", 3] },
              "Positive",
              "Negative"
            ]
          }
        ]
      }
    }
```


```
  }
]);
```

4. **Profitability Metrics**:

- PostgreSQL: Analyzed pricing and suggested adjustments based on sales performance by region.

```
WITH RegionalSales AS (
    -- Aggregate sales by region using geolocation information
    SELECT
        g.geolocation_state AS region,
        p.product_id,
        p.product_category_name,
        COUNT(oi.order_item_id) AS total_items_sold,
        SUM(oi.price) AS total_sales_value,  -- Using price from order_items table
        AVG(oi.price) AS average_price       -- Using price from order_items table
    FROM orders o
    JOIN order_items oi ON o.order_id = oi.order_id
    JOIN products p ON oi.product_id = p.product_id
    JOIN geolocation g ON CAST(SUBSTRING(o.customer_id, 1, 5) AS text) = CAST(g.geolocation_zip_code_prefix AS text)
    GROUP BY g.geolocation_state, p.product_id, p.product_category_name
),
SalesPerformance AS (
    -- Calculate performance metrics by region
    SELECT
        region,
        product_category_name,
        total_sales_value,
        total_items_sold,
        average_price,
        -- Price elasticity calculation: simple estimate based on sales volume vs. price
        (total_items_sold / NULLIF(average_price, 0)) AS price_elasticity
    FROM RegionalSales
)
SELECT
    region,
```

```
    product_category_name,
    total_sales_value,
    total_items_sold,
    average_price,
    price_elasticity
FROM SalesPerformance
ORDER BY price_elasticity DESC;  -- Products with highest price elasticity come first
```

Query | Query History | Scratch Pad

Data Output | Messages | Notifications

	region character varying (255)	product_category_name character varying (255)	total_sales_value numeric	total_items_sold bigint	average_price numeric	price_elasticity numeric
1	RS	esporte_lazer	1285.20	210	6.1200000000000000	34.3137254901960784
2	MG	esporte_lazer	5558.00	397	14.0000000000000000	28.3571428571428571
3	RS	beleza_saude	12393.00	510	24.3000000000000000	20.9876543209876543
4	MG	perfumaria	2088.20	197	10.6000000000000000	18.5849056603773585
5	SC	esporte_lazer	11375.00	455	25.0000000000000000	18.2000000000000000
6	MG	beleza_saude	4315.50	274	15.7500000000000000	17.3968253968253968
7	SP	eletronicos	3098.55	227	13.6500000000000000	16.6300366300366300
8	SP	eletronicos	2229.50	182	12.2500000000000000	14.8571428571428571
9	BA	telefonia	13096.20	438	29.9000000000000000	14.6488294314381271
10	SC	informatica_acessorios	1121.40	126	8.9000000000000000	14.1573033707865169
11	SP	bebes	3177.88	212	14.9900000000000000	14.1427618412274850
12	RJ	esporte_lazer	3376.82	218	15.4900000000000000	14.0735958683021304
13	SP	fashion_bolsas_e_acessorios	4728.51	249	18.9900000000000000	13.1121642969984202
14	SP	fashion_bolsas_e_acessorios	20548.50	515	39.9000000000000000	12.9072681704260652
15	MG	automotivo	5074.50	255	19.9000000000000000	12.8140703517587940
16	RJ	malas_acessorios	14902.30	427	34.9000000000000000	12.2349570200573066
17	MG	automotivo	4408.00	232	19.0000000000000000	12.2105263157894737
18	RS	cama_mesa_banho	14658.00	420	34.9000000000000000	12.0343839541547278
19	SC	moveis_decoracao	13965.00	399	35.0000000000000000	11.4000000000000000
20	SP	automotivo	3544.03	197	17.9900000000000000	10.9505280711506392
21	RJ	moveis_decoracao	26746.40	536	49.9000000000000000	10.7414829659318637
22	RJ	moveis_decoracao	27536.90	541	50.9000000000000000	10.6286836935166994
23	PR	telefonia	1230.88	112	10.9900000000000000	10.1910828025477707
24	MT	cool_stuff	3960.10	199	19.9000000000000000	10.0000000000000000
25	RS	beleza_saude	354.00	59	6.0000000000000000	9.8333333333333333
26	RJ	papelaria	8790.60	294	29.9000000000000000	9.8327759197324415
27	MG	audio	3761.10	189	19.9000000000000000	9.4974874371859296
28	MG	cool_stuff	1597.77	123	12.9900000000000000	9.4688221709006928

Total rows: 1000 of 1469   Query complete 00:00:02.936   Ln 37, Col 1

- MongoDB: Derived new fields like profit margins using $set.

```
db.order_items_products.updateMany(
  {},
  { $set: { estimated_profit_margin: { $subtract: ["$price", { $add: ["$freight_value", "$product_weight_g"] }] } } }
);
```

```
> db.order_items_products.updateMany(
    {},
    { $set: { estimated_profit_margin: { $subtract: ["$price", { $add: ["$freight_value", "$product_weight_g"] }] } } }
  );
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 112650,
    modifiedCount: 112650,
    upsertedCount: 0
  }
```

Strengths and Weaknesses of the Two Databases

PostgreSQL:

- **Strengths**:
  - Strong ACID compliance ensures transactional reliability.
  - Optimized for structured data with well-defined relationships.
  - Advanced query capabilities (e.g., window functions, CTEs).
- **Weaknesses**:
  - Limited scalability in distributed systems without external tools.
  - Rigid schema design is less adaptable to evolving data structures.

MongoDB:

- **Strengths**:
  - Schema-less design supports dynamic and semi-structured data.
  - Built for horizontal scalability with sharding across distributed systems.
  - Faster query performance for read-heavy operations.
- **Weaknesses**:
  - No native support for complex joins; requires denormalization.
  - Eventual consistency may lead to stale data in critical transactions.

Comparison of Five Factors for the Dataset

Factor	**Winner**	**Justification**
Performance	PostgreSQL	Handles complex joins and transactional queries efficiently, outperforming MongoDB on relational tasks.
Scalability	MongoDB	Horizontal scaling through sharding proved more effective for large datasets and distributed environments.
Flexibility	MongoDB	Schema-less design allowed easy handling of reviews and dynamic fields without altering the schema, whereas PostgreSQL required strict adherence to pre-defined schemas.
Complexity	PostgreSQL	Advanced query capabilities (e.g., window functions, subqueries) provided robust analytical tools for relational datasets.
Functionality	PostgreSQL	ACID compliance ensured strong consistency for financial operations like payment success rates and inventory updates.

Insights and Key Takeaways

- **PostgreSQL** is optimal for:
  - Financial operations requiring immediate consistency.
  - Complex relational queries involving multiple tables.
- **MongoDB** is ideal for:
  - Distributed systems with high availability needs.
  - Semi-structured and unstructured data like customer reviews.

Trade-Offs:

- Use PostgreSQL when **transactional integrity and structured data** are priorities.
- Use MongoDB when **scalability, flexibility, or high availability** are crucial.

Roles and Responsibilities

- **Developers (Kathan, Vraj, Harsh)**:
  - Implement PostgreSQL and MongoDB schemas.
  - Develop and optimize queries for scenarios like order processing and sentiment analysis.
  - Automate data imports and ensure database accuracy.
- **Documentation (Devarsh, Devanshi, Divy)**:
  - Create guides, project proposals, and presentations.
  - Document dataset structure, challenges, and findings.
  - Compile performance results and project insights.
- **Research (Kathan, Harsh, Vraj, Devarsh)**:
  - Analyze dataset suitability for normalization/denormalization.
  - Define metrics and evaluate database trade-offs.
  - Align implementations with real-world e-commerce scenarios.

Conclusion

This project highlights the trade-offs between PostgreSQL and MongoDB:

- **PostgreSQL** excels in **structured, transactional systems**, prioritizing data integrity and advanced analytics.
- **MongoDB** shines in **distributed, semi-structured environments**, offering flexibility and scalability.

Our findings emphasize that the optimal choice depends on specific use cases, making both databases valuable tools for modern e-commerce platforms.