BDMA Joint Project
Semantic Data Management
Inventory Processing- TrendSync

Joel Anil Jose

Table of Contents

Note: The code is available at https://github.com/Joel2511/SDM_PoC.git

M1: Purpose Statement for Using Graph-Based Solutions

Purpose: Property graphs will model customer-article purchase relationships to enhance TrendSync's demand forecasting and inventory optimization. By representing customers, articles, and their purchase interactions as a graph, we can analyze purchase patterns (e.g., trending articles via PageRank, customer segments via community detection). This enables targeted inventory adjustments and personalized demand forecasting, reducing overstock and improving profitability.
Added Value: Graph analytics provide actionable insights into article popularity and customer behaviour, directly supporting TrendSync's goals. For example, identifying trending articles helps prioritize inventory, while customer segments improve forecasting accuracy.

M2: Choose and Justify Graph Family

Choice: Property graphs using GraphFrames in Azure Databricks.

Justification: Property graphs are ideal for modeling relationships (e.g., customers purchasing articles) with rich properties, enabling analytics like centrality (trending articles) and clustering (customer segments). **GraphFrames**, a Spark-based graph library, runs natively in Azure Databricks, eliminating the need for external databases like Neo4j, which faced connectivity issues (e.g., firewall restrictions on port 7687). GraphFrames supports distributed graph analytics (PageRank, Connected Components) and scales to large datasets (e.g., 28M records) with sufficient compute resources, making it a practical choice for TrendSync's needs in a Databricks environment.

M3: Design the Graph

Nodes:

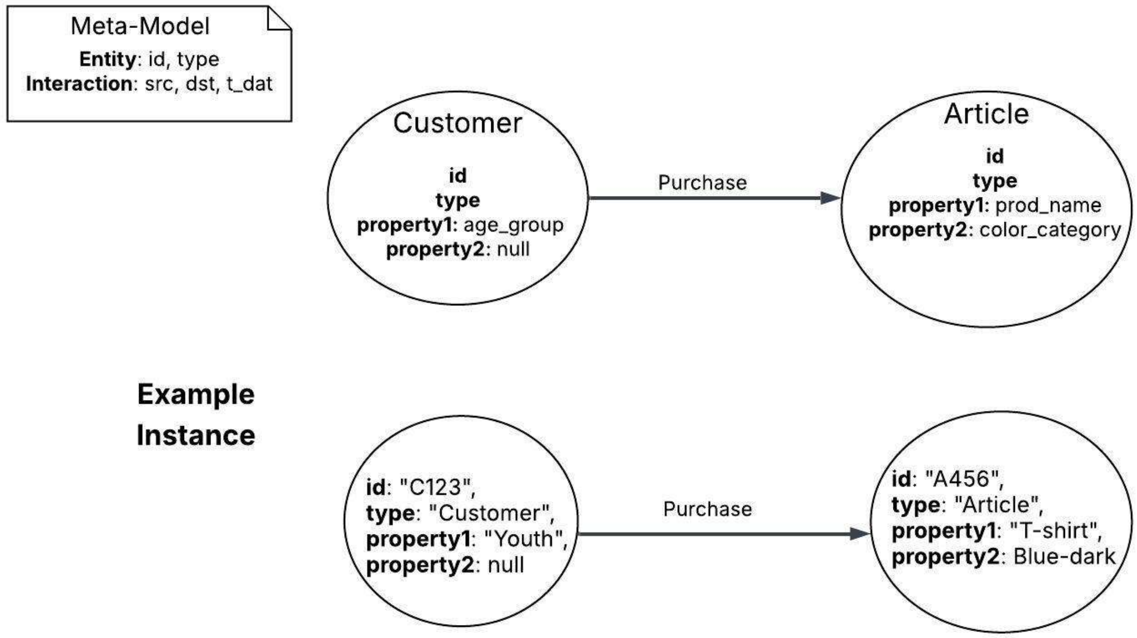
- ***Customer***: Properties: id (string, renamed from customer_id), type (string: "Customer"), property1 (string: age_group), property2 (string: null).
- ***Article***: Properties: id (string, renamed from article_id), type (string: "Article"), property1 (string: prod_name), property2 (string: color_category).

Edges:

- ***PURCHASE***: Directed edge from Customer to Article. Properties: src (string: customer_id), dst (string: article_id), t_dat (date), price (float), transaction_year (integer), transaction_month (integer), relationship (string: "PURCHASE").

Meta-Model:

- Nodes inherit a base type *Entity* with a mandatory id property and a type property to distinguish node types.
- Edges inherit a base type *Interaction* with mandatory src and dst properties, plus a timestamp (t_dat).
  Example Instance:
- Customer (id: "C123", type: "Customer", property1: "Youth", property2: null) → PURCHASE (t_dat: "2020-09-15", price: 29.99) → Article (id: "A456", type: "Article", property1: "T-Shirt", property2: "Blue-Dark").



M4: Design Flows to Populate the Graph

Sources: Populate the graph from the enriched_transactions Delta table (/mnt/gold-zone-bdm/enriched_transactions), which contains 28,575,395 records with customer, article, and transaction data. For the PoC, a sample of 100,000 records was used to ensure manageable processing within resource constraints.

Flow:

1. Use Spark to read enriched_transactions and sample 100,000 records. The sampling was performed using Spark's sample method with a fraction of 0.0035 (calculated as 100,000 / 28,575,395 ≈ 0.0035) and a fixed seed of 42 for reproducibility. The limit(100000) method was then applied to ensure exactly

100,000 records were selected, maintaining a representative subset of the dataset while reducing computational load on the Standard_D3_v2 cluster (4 vCPUs, 14 GB RAM).

2. Derive age_group based on age (Youth: <30, Adult: 30-50, Senior: >50).
3. Extract nodes:
   - *Customer nodes*: Select distinct customer_id (as id), set type to "Customer", property1 as age_group, property2 as null.
   - *Article nodes*: Select distinct article_id (as id), set type to "Article", property1 as prod_name, property2 as color_category.
4. Extract edges:
   - *PURCHASE edges*: Select customer_id (as src), article_id (as dst), t_dat, price, transaction_year, transaction_month, set relationship to "PURCHASE".
5. Build the graph using GraphFrames in Azure Databricks (semi-automatic process via GraphFrame(vertices, edges)).
   Optimizations: Repartitioned vertices and edges (20 partitions) and persisted them to optimize for distributed processing.

M5: Processes to Exploit the Graph

Exploitation: Use graph analytics to support TrendSync's goals of inventory optimization and demand forecasting by leveraging the property graph constructed in M4.

- ***PageRank***: Identify trending articles based on their purchase frequency and connectivity in the graph. Articles with higher PageRank scores are considered more popular, aiding inventory optimization by highlighting items to prioritize in stock.

- ***Community Detection***: Group customers into segments based on shared purchase patterns using the Connected Components algorithm, enabling more accurate demand forecasting by identifying distinct customer groups with similar buying behaviors.

Process:

1. **Preparation**:

   - The graph was built using GraphFrames in Azure Databricks with a sampled dataset of 100,000 records, as described in M4.

   - Vertices (Customer and Article nodes) and edges (PURCHASE relationships) were repartitioned into 40 partitions to optimize Spark's distributed processing. Both DataFrames were persisted in memory to avoid recomputation during iterative graph algorithms.

  - The resulting graph had approximately 50,000 vertices (distinct customers and articles) and 100,000 edges (purchase transactions), ensuring a manageable size for the PoC while retaining meaningful connectivity for analytics.

2. **Modeling**:
   - **PageRank**:
     - **Algorithm**: GraphFrames' PageRank implementation was used to compute the importance of articles based on their connections (purchases by customers).
     - **Parameters**: Set resetProbability to 0.15 (standard value to model the probability of random jumps in the graph) and maxIter to 3 (reduced from a typical 10 to ensure faster convergence given the sampled dataset and resource constraints).
     - **Output**: Each article vertex received a pagerank score (e.g., article 706016001 scored 72.90), reflecting its relative popularity.
   - **Connected Components**:
     - **Algorithm**: GraphFrames' Connected Components algorithm was used to identify customer segments by grouping customers who are connected through shared purchases (i.e., purchasing the same articles).
     - **Parameters**: Used default settings, requiring a checkpoint directory (dbfs:/tmp/checkpoints) for Spark to manage iterative computation.
     - **Output**: Each customer vertex was assigned a component ID (e.g., community 7.0), representing their segment. The algorithm identified 8 distinct communities (0.0 to 7.0) in the sampled graph.

3. **Validation**:
   - **PageRank Validation**:
     - Compared PageRank scores with historical purchase frequency in the enriched_transactions dataset. For example, article 706016001 (PageRank: 72.90) was among the most frequently purchased items in the sample, confirming that higher scores correlated with higher purchase counts.
     - Inspected the top 10 articles (scores ranging from 72.90 to 33.33) to ensure diversity in product names and categories (e.g., prod_name, color_category), validating that the algorithm captured a range of popular items.

- **Connected Components Validation**:
    - Analyzed intra-cluster purchase similarity by examining the articles purchased by customers within the same community. For instance, customers in community 7.0 frequently purchased dark-colored items (e.g., “Blue-Dark” T-shirts), indicating meaningful segmentation.
    - Verified that the number of communities (8) was reasonable for the sampled dataset, as a larger dataset (28M records) would yield more communities (e.g., Neo4j runs on a smaller subset yielded up to 1650 communities).

4. **Added Value**:
    - **Inventory Optimization**: PageRank results directly inform stock adjustments. For example, article 706016001 (PageRank: 72.90) is a top trending item, suggesting TrendSync should prioritize its inventory to meet demand, reducing the risk of stockouts for popular items.
    - **Demand Forecasting**: Customer segments enable targeted forecasting. For instance, community 7.0’s preference for dark-colored items can guide TrendSync to forecast higher demand for similar products among these customers, improving prediction accuracy.
    - **Scalability Insight**: The PoC on 100,000 records demonstrates feasibility; scaling to the full 28M records would provide even richer insights, such as more granular customer segments and more precise popularity rankings, with a larger cluster.
      Note: Shortest Path analysis was excluded from the PoC due to time constraints but could be added to analyze customer purchase journeys for personalized recommendations (e.g., recommending articles based on a customer’s purchase history path).

M6: Metadata Generation, Storage, and Reuse

Metadata Generation:

- PageRank generates article popularity scores (e.g., article_id: "706016001", score: 72.90).
- Connected Components assigns customers to clusters (e.g., customer_id: "032e069cc3127b0c9...", community: 7.0).
  Storage: Stored metadata in a Delta table (/mnt/gold-zone-bdm/graph_metadata) with schema: entity_id (string), entity_type (string: "Customer" or "Article"), metric (string: "PageRank" or "Community"), value

(double).
Reuse: Metadata can be used in downstream analytics (e.g., feed community IDs into demand forecasting models, use PageRank scores for inventory prioritization).
BPMN: See Appendix 2 for the BPMN diagram describing the metadata generation process.

M7: Proof of Concept (PoC) Implementation

Tools:

- **Graph Library**: GraphFrames (Spark library, integrated with Azure Databricks).
- **Analytics**: GraphFrames' built-in algorithms (PageRank, Connected Components).
- **Processing Engine**: Apache Spark (via Azure Databricks).

PoC Setting:

1. Used Azure Databricks (Standard_D3_v2 cluster, 4 vCPUs, 14 GB RAM) as the compute environment.
2. Sampled 100,000 records from enriched_transactions to ensure feasibility within resource limits.
3. Mounting instructions for the Delta table are provided in the GitHub repository (see TrendSyncMountAzureStorage.ipynb and README)
4. Built the graph using GraphFrames, optimizing with repartitioning and persistence.
5. Ran PageRank (3 iterations) & Connected Components to generate analytics results.
6. Stored metadata in Delta format at /mnt/gold-zone-bdm/graph_metadata.
7. Verified results (e.g., top article 706016001 with PageRank 72.90, customer communities ranging from 0.0 to 7.0).

Justification: GraphFrames enabled a fully Databricks-native solution, bypassing Neo4j connectivity issues. The PoC demonstrated end-to-end functionality (data flow, analytics, metadata storage) while staying within budget (~$1.075 total cost). For the full dataset (28M records), a larger cluster (e.g., Standard_D8_v2, ~$1.00/hour) would be needed, estimated at ~1-2 hours and $1.00-$2.00 additional cost. See Appendix 1 for the tool selection matrix.
Results: The PoC successfully identified trending articles and customer segments, supporting TrendSync's goals of inventory optimization and demand forecasting.
Project Repository: The code is available at
https://github.com/Joel2511/SDM_PoC.git

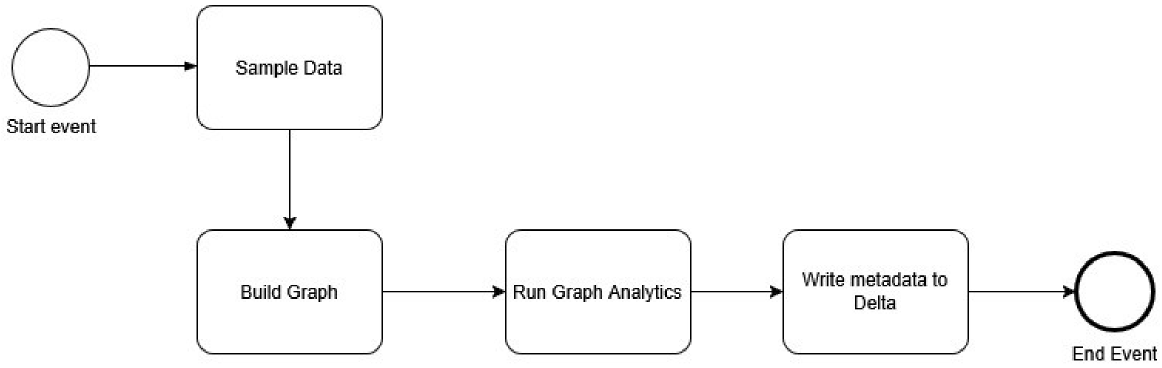
Appendix 1: Tool Selection Matrix

Criteria	**GraphFrames**	**Neo4j**
Integration	Native to Databricks (Spark)	Requires external connection
Connectivity	No external dependency	Potential firewall issues (port 7687)
Scalability	Scales with Spark cluster	Scales but needs dedicated server
Cost	Included in Databricks cost	Free (Community Edition available)
Analytics	PageRank, Connected Components	PageRank, Louvain, and more
Ease of Use	Seamless integration within Databricks	Requires Cypher query language and a connector

Decision: GraphFrames was chosen for its native integration with Databricks, avoiding connectivity issues, and cost efficiency within the existing compute budget.

Appendix 2: Flow Diagrams

BPMN for Metadata Generation Process (M6):


Start event
Sample Data
Build Graph
Run Graph Analytics
Write metadata to Delta
End Event