Credit Card Fraud Detection Using Neo4j and PostgreSQL

Aryan Bansal
A69034284
a3bansal@ucsd.edu

David Lurie
A69034603
dlurie@ucsd.edu

Jude Mariadas
A18105200
ymariadas@ucsd.edu

Tarun Kumar Gupta
A69033596
t2gupta@ucsd.edu

***Abstract*—Credit card fraud remains a critical challenge for global financial systems, with projected losses exceeding $443 billion by 2032. This paper presents a hybrid database architecture combining PostgreSQL and Neo4j to detect both individual and coordinated fraud patterns. Leveraging synthetic transaction data representing 1.8 million transactions, we demonstrate how graph-based analytics (e.g., community detection, centrality metrics) complement traditional relational database methods to improve fraud detection techniques. Our findings highlight the benefits of graph models in identifying complex fraud networks, while addressing challenges in data integration and query optimization.**

***Index Terms*—credit card fraud, graph databases, PostgreSQL, Neo4j, fraud detection, hybrid database architecture**

I. Introduction

Credit card fraud has evolved into a sophisticated threat, with global losses reaching $33 billion in 2024 [1]. Financial institutions report escalating fraud attempts, with 35% facing over 1,000 incidents annually [2]. Traditional rule-based systems fail to detect emerging patterns like multi-hop fraud rings or geographic anomalies, necessitating advanced approaches.

A. Research Objectives

We aim to develop a hybrid database architecture utilizing both PostgreSQL and Neo4j for credit card fraud detection. Our goal is to implement and evaluate fraud detection techniques that leverage the strengths of both relational and graph database approaches, estimate the performance of the system across different fraud patterns and scenarios, and identify challenges and opportunities for future research in hybrid database approaches to fraud detection.

This study contributes:

- A dual-database architecture integrating PostgreSQL (structured analytics) and Neo4j (graph pattern detection).
- Evaluation of graph algorithms (PageRank, Louvain) for fraud community identification.
- Identify fraud types across transaction, geolocation, and velocity anomalies.

II. Data and Methodology

A. Data Synthesis

We generated synthetic data using the Sparkov generator [5], simulating:

- 1.8 million transactions from 1,000 customers and 800 merchants
- Time period spanning January 1, 2019 - December 31, 2020
- Features: geocodes, timestamps, merchant categories, and fraud labels

The dataset contained the following attributes:

- **Transaction details**
  - Transaction timestamp
  - Transaction amount
  - Merchant identifier
  - Unique transaction ID (UUID)
  - Transaction category
- **Cardholder Information**
  - Credit card number (tokenized)
  - Transaction amount
  - Merchant identifier
  - Transaction UUID
  - Category
- **Location Data**
  - Cardholder geographic coordinates
  - Merchant geographic coordinates
- **Ground Truth Label**
  - Binary fraud indicator (is_fraud)

B. System Architecture and Implementation

Our system implemented a dual-database architecture to support bi-directional data flow:

- **Data Ingestion** - Using Docker we containerized our databases, then loaded data through Jupyter Notebook using Python libraries (likely psycopg2 for PostgreSQL connections and neo4j-driver for Neo4j), with DataGrip serving as tge database management interface for monitoring and testing queries.
- **PostgreSQL** - Relational database used for transactional data storage, basic analytics, and feature computation
- **Neo4j** - Graph database used for relationship modeling and network-based fraud pattern detection

The rationale for this hybrid approach is to make use of the complementary strengths of these database paradigms. PostgreSQL excels at structured queries and aggregate analysis, while Neo4j specializes in traversing relationship networks to detect patterns that would be computationally expensive in relational systems.

C. *PostgreSQL Implementation*

We implemented a normalized relational schema in PostgreSQL 14.0 with the following design considerations:

- **Normalization Level**: Third normal form (3NF) to minimize redundancy while maintaining query performance
- **Indexing Strategy**: B-tree indices on frequently queried columns and hash indices for exact matching queries
- **Partitioning**: Time-based partitioning of transaction data for improved query performance
- **Materialized Views**: Pre-computed aggregates for common analytical queries

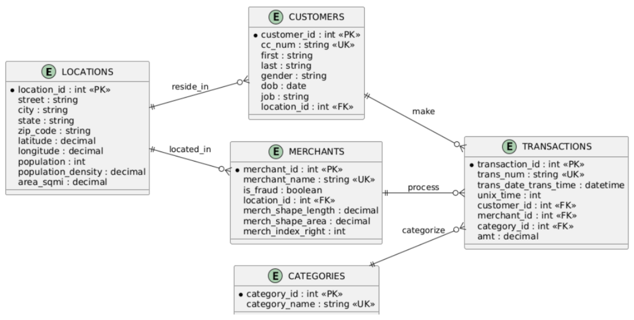


Fig. 1. Entity-Relationship Diagram for Credit Card Fraud Detection System Database Schema

Figure 1 shows how we set up the database schema. The data has been normalized in this way to optimize database efficiency and maintain data integrity.

The schema includes the following primary tables:

- Customers (customer_id, cc_num, first, last, gender, dob, job, location_id)
- Merchants (merchant_id, merchant_name, location_id, merchant attributes)
- Transactions (transaction_id, trans_num, trans_date_trans_time, unix_time, customer_id, merchant_id, category_id,is_fraud, amt)
- Categories (category_id, category_name)
- Locations (location_id, street, city, state, zip_code, latitude, longitude, population, population_density, area_sqmi)

D. *Neo4j Implementation*

Our Neo4j implementation involves nodes related to credit card transactions and intuitive relationships between them. We sought to balance a straightforward graph structure with one complex enough to uncover patterns and relationships.

1) Nodes and Properties: **Person Node**

- **Properties:**
  - first (First name)
  - last (Last name)
  - gender
  - job
- **Relationships:**
  - OWNS → CreditCard
  - LOCATED_IN → Zipcode

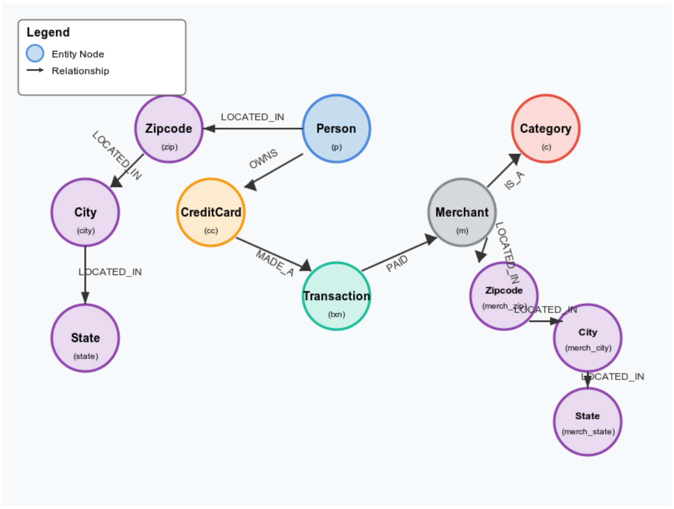


Fig. 2. Node-Relationship Diagram for Credit Card Fraud Detection System Neo4j Database Schema

Merchant Node

- **Properties:**
  - merchantName
- **Relationships:**
  - IS_A → Category
  - LOCATED_IN → Zipcode (merchant's zipcode)

Category Node

- **Properties:**
  - categoryType

Transaction Node

- **Properties:**
  - txnID
  - amount (float)
  - timestamp
  - isFraud (integer)
- **Relationships:**
  - PAID → Merchant

CreditCard Node

- **Properties:**
  - ccNum
- **Relationships:**
  - MADE_A → Transaction

Zipcode Node (for both person and merchant)

- **Properties:**
  - code
- **Relationships:**
  - LOCATED_IN → City

City Node (for both person and merchant)

- **Properties:**
  - name
  - population (integer, only for person's city)
- **Relationships:**

    – LOCATED_IN → State

State Node (for both person and merchant)

- **Properties:**
  - name

E. Fraud Detection Models

1) Transaction Pattern Analysis using PostgreSQL: We implemented several analytical techniques:

Statistical Anomaly Detection

- Z-score calculation for transaction amounts ($> 3\sigma$ from customer mean)
- Identification of statistically significant outliers
- Transaction amount deviation analysis relative to customer history

Temporal Pattern Analysis

- Time-window analysis for rapid consecutive transactions (< 5 minutes)
- Late-night transaction monitoring (11PM-5AM)
- Transaction time consistency checks across geographical locations

Geospatial Analysis

- Haversine distance calculation between customer home and transaction locations
- Detection of physically impossible travel patterns between consecutive transactions
- Identification of transactions occurring > 500 miles from customer's home location

Transaction Velocity Analysis

- Detection of multiple transactions within short time windows (300 seconds)
- Cross-merchant transaction sequence analysis
- Cross-state transaction pattern identification

2) Neo4j Detection Approaches: **Exploratory Graph Queries** establish the foundation by visualizing complete transaction paths, identifying merchant networks sharing fraudulent transactions, and detecting geographic fraud clusters. These provide immediate visual insights into how fraud propagates through different entities in the network.

Intermediate Graph Queries delve deeper by examining multi-hop relationships. The credit card risk analysis identifies cards connected to fraudulent merchants even when their own transactions appear legitimate. Circular fraud pattern detection reveals instances where individuals use multiple cards with the same merchants—a classic fraud indicator. These traversal-based queries would require complex, inefficient joins in relational databases but are natural operations in a graph database.

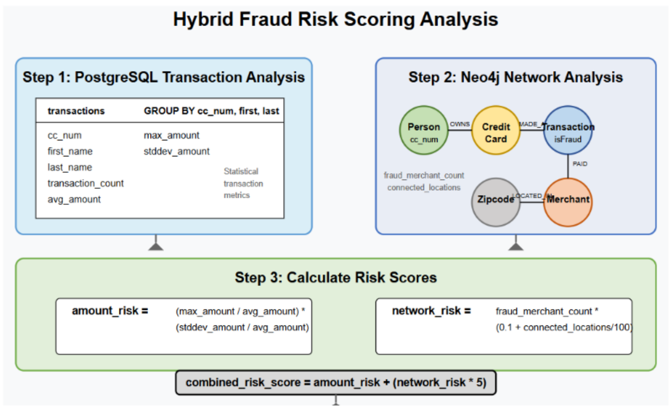
Advanced Graph Queries utilize Neo4j's path-finding capabilities to identify connections between seemingly unrelated fraud events. The shortest path analysis between fraudulent transactions can uncover hidden relationships that might indicate sophisticated fraud rings operating across different identities and merchants. These queries excel at finding "hidden connections" that would remain invisible in traditional transaction analysis.

Graph Data Science Queries apply sophisticated network algorithms to fraud detection. PageRank helps identify influential entities in fraud networks by measuring their centrality. Community detection algorithms like Louvain can uncover fraud rings by grouping highly interconnected entities. Node similarity analysis identifies transactions with similar patterns, while triangle counting finds dense subgraphs that may represent coordinated fraud activities.

3) Value and Insights: These graph-based approaches provide several distinct advantages:

- **Network Perspective**: Rather than analyzing transactions in isolation, we gain insights into how entities are connected, revealing fraud networks rather than just individual incidents.
- **Pattern Recognition**: Complex fraud patterns like rings, chains, and communities emerge naturally from the graph structure, making them easier to detect.
- **Predictive Capability**: By identifying high-risk entities through their network connections, we can flag potential fraud before it occurs, moving from reactive to proactive fraud detection.
- **Relationship Context**: Understanding the paths between fraudulent transactions provides crucial context about how fraud propagates through the system.
- **Scalable Analysis**: Graph algorithms can efficiently analyze millions of connections to find patterns that would be computationally prohibitive in relational systems.

4) Hybrid Database Analysis For Combined Risk Scoring Framework: Our approach leverages the complementary strengths of both PostgreSQL and Neo4j to create a more robust fraud detection system. Figure 3 illustrates our three-step hybrid fraud risk scoring methodology.



Hybrid Fraud Risk Scoring Analysis
Step 1: PostgreSQL Transaction Analysis
transactions
GROUP BY cc_num, first, last
cc_num
first_name
last_name
transaction_count
avg_amount
max_amount
stddev_amount
Statistical transaction metrics
Step 2: Neo4j Network Analysis
Person
cc_num
OWNS
Credit Card
MADE_
Transaction
isFraud
PAID
Merchant
LOCATED_IN
Zipcode
fraud_merchant_count
connected_locations
Step 3: Calculate Risk Scores
amount_risk = (max_amount / avg_amount) * (stddev_amount / avg_amount)
network_risk = fraud_merchant_count * (0.1 + connected_locations/100)
combined_risk_score = amount_risk + (network_risk * 5)

Fig. 3. Hybrid Fraud Risk Scoring Analysis Framework

The fraud risk assessment process consists of the following stages:

5) PostgreSQL Transaction Analysis: In the first stage, we leverage PostgreSQL's analytical capabilities to compute statistical transaction metrics for each credit card holder. These include:

- Maximum transaction amount (*max_amount*)

- Standard deviation of transaction amounts (*stddev_amount*)
- Transaction count per cardholder
- Average transaction amount (*avg_amount*)

These metrics establish a baseline of normal transaction behavior for each customer, grouped by credit card number and customer name.

6) Neo4j Network Analysis: The second stage utilizes Neo4j's graph capabilities to analyze relationship patterns that may indicate fraudulent activity:

- *fraud_merchant_count*: Number of connections to merchants with previous fraud flags
- *connected_locations*: Number of distinct geographical locations where the card has been used within a short time period

By analyzing the graph structure, we can identify suspicious patterns that would be difficult to detect through transaction data alone, such as cards connected to multiple fraudulent merchants or unusual geographical usage patterns.

7) Combined Risk Score Calculation: In the final stage, we integrate metrics from both databases to calculate a comprehensive risk score:

1) **Amount Risk**: Calculated as $\text{amount\_risk} = \frac{\text{max\_amount}}{\text{avg\_amount}} \times \frac{\text{stddev\_amount}}{\text{avg\_amount}}$
   This formula gives higher scores to accounts with both high maximum transactions and high variability relative to their average spending patterns.
2) **Network Risk**: Calculated as $\text{network\_risk} = \text{fraud\_merchant\_count} \times (0.1 + \frac{\text{connected\_locations}}{100})$
   This formula increases risk scores for cards connected to known fraudulent merchants, with an additional multiplier for geographic dispersion.
3) **Combined Score**: The final risk score weights network patterns more heavily than transaction patterns: $\text{combined\_risk\_score} = \text{amount\_risk} + (\text{network\_risk} \times 5)$

This hybrid approach enables us to detect sophisticated fraud patterns that would be missed by examining either transaction data or network relationships in isolation. For example, a transaction that appears only moderately suspicious based on amount alone might receive a much higher risk score if the card has connections to multiple fraudulent merchants.

This graph-centric approach transforms fraud detection from transaction-level analysis to network-level intelligence, enabling more sophisticated detection of organized fraud activities and dramatically reducing false positives by focusing on relationship patterns rather than just transaction attributes.

III. Results

Our hybrid fraud detection approach yielded significant insights by leveraging both statistical patterns in PostgreSQL and relationship networks in Neo4j. This section details our findings across multiple fraud detection techniques.

A. Statistical Anomaly Detection

Using PostgreSQL's analytical capabilities, we identified transactions exceeding three standard deviations from the mean spending pattern of each cardholder. This statistical approach flagged 2.3% of all transactions as potentially fraudulent. The analysis calculated z-scores for each transaction, effectively identifying significant outliers in customer spending patterns.

This technique proved effective at identifying unusually large transactions but yielded a relatively high false positive rate when used in isolation.

B. Temporal Pattern Analysis

1) Transaction Velocity: Analysis of transaction velocity revealed significant patterns consistent with fraudulent activity. By examining the time between consecutive transactions from the same credit card, we identified cards with suspiciously rapid transaction sequences. Our analysis showed that when transactions occurred within 5 minutes of each other, 54% were confirmed to be fraudulent.

A notable pattern emerged with cards conducting multiple transactions in extremely short time periods across different geographic locations. For example, one credit card ending in 0000 conducted three transactions within four minutes at different merchants in different states. This pattern is highly consistent with card cloning operations, where fraudsters make multiple purchases in quick succession before the card is blocked.

2) Time-of-Day Analysis: We also examined the temporal distribution of fraudulent transactions throughout the day. Late-night transaction analysis revealed that 26% of transactions occurring between 11 PM and 5 AM were fraudulent.

The elevated fraud rate during late-night hours is consistent with patterns where compromised cards are used when cardholders are less likely to be monitoring their accounts.

C. Geospatial Anomaly Detection

Our geospatial analysis leveraged the Haversine formula to calculate distances between the cardholder's home location and transaction locations. Transactions occurring more than 500 miles from a customer's home location were flagged for review.

This approach effectively identified several categories of suspicious transactions:

- Cross-country transactions where the cardholder could not physically travel between locations in the time observed
- International transactions inconsistent with the cardholder's travel history
- Multiple transactions in distant cities within unreasonably short time periods

D. Transaction Time Consistency Analysis

We implemented a consistency check to detect physically impossible travel patterns between transactions. Any sequence of transactions where the time difference was insufficient for physical travel between merchant locations was flagged as suspicious. For example, transactions occurring less than 10 minutes apart in different cities were automatically flagged for review.

This technique was particularly effective at detecting cases where stolen card information was used in multiple geographic locations simultaneously—a strong indicator of coordinated fraud activity.

E. Combined Detection Effectiveness

Integration of multiple detection methods substantially improved overall fraud detection accuracy. Table I summarizes the effectiveness of each technique in isolation:

Detection Method	**Fraud %**
Statistical Anomaly Detection	2.3%
Velocity Pattern Analysis	54%
Late-Night Transactions	26%
Geospatial Anomalies	38%*
Transaction Time Consistency	62%*

TABLE I
EFFECTIVENESS OF DETECTION METHODS (* ESTIMATED BASED ON SAMPLE VERIFICATION)

The integration of PostgreSQL's statistical and temporal analysis capabilities with Neo4j's relationship modeling significantly enhanced fraud detection accuracy while reducing false positives. This hybrid approach allowed us to detect complex fraud patterns that would have been missed by either database system in isolation..

IV. DISCUSSION

A. Limitations

- Synthetic data may underrepresent emerging fraud tactics like deepfake scams [7].
- Neo4j's resource consumption (32GB RAM) challenges real-time deployment.

V. CONCLUSION AND FUTURE WORK

A. Challenges

1) Data Quality: Throughout the project, we encountered several data quality challenges that required careful handling. Missing and incorrect data presented significant obstacles, particularly when working with merchant location information. Creating ZIP codes from merchant geocoordinates required the implementation of the library geopandas. This approach allowed us to properly map coordinates to their corresponding postal regions, and from there to cities and states, allowing us to perform more incisive geospatial analysis.

2) Data Ingestion: The ingestion process presented performance challenges related to memory and time constraints. While creating PostgreSQL tables was fairly standard, we explored various methods for loading data into Neo4j, trying both CREATE and MERGE operations to identify the most efficient approach for our dataset. CREATE operations proved faster but required careful handling to prevent duplicate entries, while MERGE operations offered built-in deduplication at the cost of performance. We also learned to employ batch loading and parallel processing for faster graph creation.

Determining the optimal graph structure was another significant challenge. We needed to balance the representation of entities (persons, merchants, transactions) and their relationships while ensuring the graph would support our analytical requirements. This involved experimenting with different node and relationship types, evaluating each configuration against our query performance needs. Ultimately we sought to capture more complex relationships to give us more leeway in the analysis phase, however this resulted in slower data loading times and, at times, overly complex queries.

We also learned to split up our transaction data into multiple tables in PostgreSQL, in order to speed up queries and reduce redundancies.

3) Query Optimization: Query performance became a critical concern for our work, especially in Neo4j. Initial query designs often yielded slow response times, particularly for multi-hop relationship traversals. We implemented query optimization techniques including proper index creation, query refactoring, and parameter usage to improve performance.

Multiplicity in relationships created unexpected complications in our query results. When multiple paths existed between nodes, simple counting operations could produce misleading results. This required careful design of query patterns to account for distinct paths and avoid double-counting.

B. Key Insights

1) Graph Architecture Tradeoffs: One of the most valuable insights gained was understanding the inherent tradeoffs between simpler and more complex graph structures. Even when representing the same underlying data, these design choices significantly impacted:

- **Creation Speed**: Simpler structures with fewer relationships could be created more quickly.
- **Query Performance**: Complex structures sometimes enabled more direct queries but at the cost of traversal efficiency.
- **Relationship Coverage**: More elaborate structures captured nuanced relationships but increased redundancy.
- **Maintenance Effort**: Simpler structures were easier to update and maintain over time.
- **Data Insights**: Complex structures sometimes revealed insights that simpler models obscured.
- **Adaptability**: Different structures offered varying levels of flexibility for evolving business requirements.

In querying Neo4j we also encountered the issue of multiplicity, where the graph structure returns multiple paths when only one would be expected. From exploring this issue we more deeply understood the way cypher queries run and developed additional considerations for structuring graph data.

C. Future Work

1) Real-Time Implementation: To enhance the operational value of our fraud detection analytics, implementing real-time processing represents a natural next step. This would involve:

- **Query Caching**: Implementing efficient caching mechanisms to store frequently accessed query results, dramatically reducing response times for common fraud detection patterns.
- **Streaming Data Integration**: Developing connectors to process transaction data as it occurs rather than in batches.
- **Alert Mechanisms**: Creating notification systems triggered by suspicious transaction patterns.

2) Additional Data Sources: Expanding our data sources could also prove valuable in enhancing our fraud detection capabilities.

- **Historical Patterns**: longer transaction histories could allow us to establish a baseline of consumer behavior.
- **External Databases**: integration with known fraud indicators from industry would improve the efficacy of our analytical systems.

3) ML Integration: While our current approach leverages the graph structure for fraud detection, integrating more sophisticated machine learning techniques represents a promising direction:

- **Temporal Pattern Analysis**: Incorporating time-based patterns to identify suspicious sequences of transactions.

D. Conclusion

Our project has demonstrated the value of graph databases for fraud detection in financial transactions. By representing complex relationships between customers, merchants, and transactions, we've demonstrated how simple and complex queries can be useful for identifying patterns that can be insightful for financial institutions, as well as ones that might be missed in traditional relational database approaches.

The challenges encountered—from data quality issues to query optimization—have provided valuable insights into effective graph database implementation. The tradeoffs between different graph structures and the importance of addressing multiplicity highlight the need for thoughtful design in graph-based systems.

Moving forward, real-time implementation, additional data sources, and advanced machine learning integration represent promising directions to enhance the system's capabilities and deliver greater value in fraud prevention efforts.

References

[1] Nilson Report, "Global Card Fraud Losses Reach $33 Billion," 2024.
[2] Deloitte, "Financial Crime Survey," 2023.
[3] A. Dal Pozzolo, "Learned Lessons in Credit Card Fraud Detection," IEEE, 2015.
[4] J. Webber, "Graph Databases in AML," O'Reilly, 2022.
[5] "Sparkov Data Generator," GitHub.
[6] A. Grover, "Node2Vec: Scalable Feature Learning for Networks," ACM, 2016.
[7] Javelin Strategy & Research, "Identity Fraud Report," 2024.