SEMANTIC DATA MANAGEMENT

BDMA Joint Project

AmiGo's Business Project - SDM Part

June 11, 2024

Authors:
Dilbar Isakova
Shofiyyah Nadhiroh

Professor:
Oscar Romero
Anna Queralt

1 Purpose Statement (M1)

All files, scrupts, visuals can be found and reviewed on this Github Repository.

Our project employs a graph-based solution to construct a social network within the "AmiGo" app, enabling enhanced event recommendations and social interactions. By utilizing property graphs, the project aims to identify patterns and relationships among users and events, facilitating more accurate and relevant event suggestions. This approach benefits the project by providing a flexible and scaleable data model that can efficiently handle complex queries and enhance user engagement through personalized recommendations.

In our Big Data Management strategy, we previously adopted Spark MLib to train content-based and collaborative filtering models. However, we encountered several challenges. By adopting property graphs, these issues can be addressed effectively. The following benefits can be achieved through the use of property graphs:

1. **Enhanced Discovery**: By leveraging relationships such as "join," "like," and "share," the system can uncover indirect connections between users and events. Even if the events themselves are not similar, recommendations can still be made based on social connections.
2. **Improved Cold Start Handling**: New users and events can be better integrated into the recommendation system by leveraging the existing social network. For example, new users can receive recommendations based on the activities of their friends or similar users.
3. **Better Scalability and Performance**: As the number of users and events grows, the computational complexity of generating recommendations increases significantly, potentially leading to performance issues. Property graphs are designed to handle complex queries efficiently, enabling scalable and performance recommendations even as the user and event base grows.

2 Graph-Based Solutions in the Project (M2)

For this project, we will use property graphs. Property graphs are most appropriate for this problem due to their flexibility in modeling complex relationships and attributes directly on nodes and edges. This allows for a more intuitive representation of the social network and event participation data. Specifically, property graphs support the following features essential for our recommendation algorithm:

1. **Nodes**: Represent entities such as Users, Events, and Locations, and can store properties like user ID, event ID, and location coordinates.
2. **Edges**: Represent relationships between nodes, such as join, like, and share, and can also store properties.
3. **Efficient Traversal**: Facilitates the traversal of relationships to find patterns such as common event participation among users.

Add to all, using property graphs enables us to:

- **Integrate diverse data sources**: Merging data from user profiles, event details, and social interactions to form a comprehensive recommendation system.

- **Facilitate real-time recommendations**: Using graph algorithms to deliver instant event suggestions based on live user activity and interactions.
- **Support dynamic schema**: Allowing the graph structure to adapt as new types of relationships and attributes are discovered, without requiring extensive database schema changes.

These features collectively enhance the AmiGo app's robustness and responsiveness, ensuring a superior user experience through tailored event recommendations and social networking functionalities.

3 Graph Schema Design (M3)

The graph schema design is based on the provided diagram, which includes entities such as **User**, **Event**, **Location**, **Location_type**, and **Event_type**. The relationships between these entities are represented with arrows, indicating the interactions and associations among them.

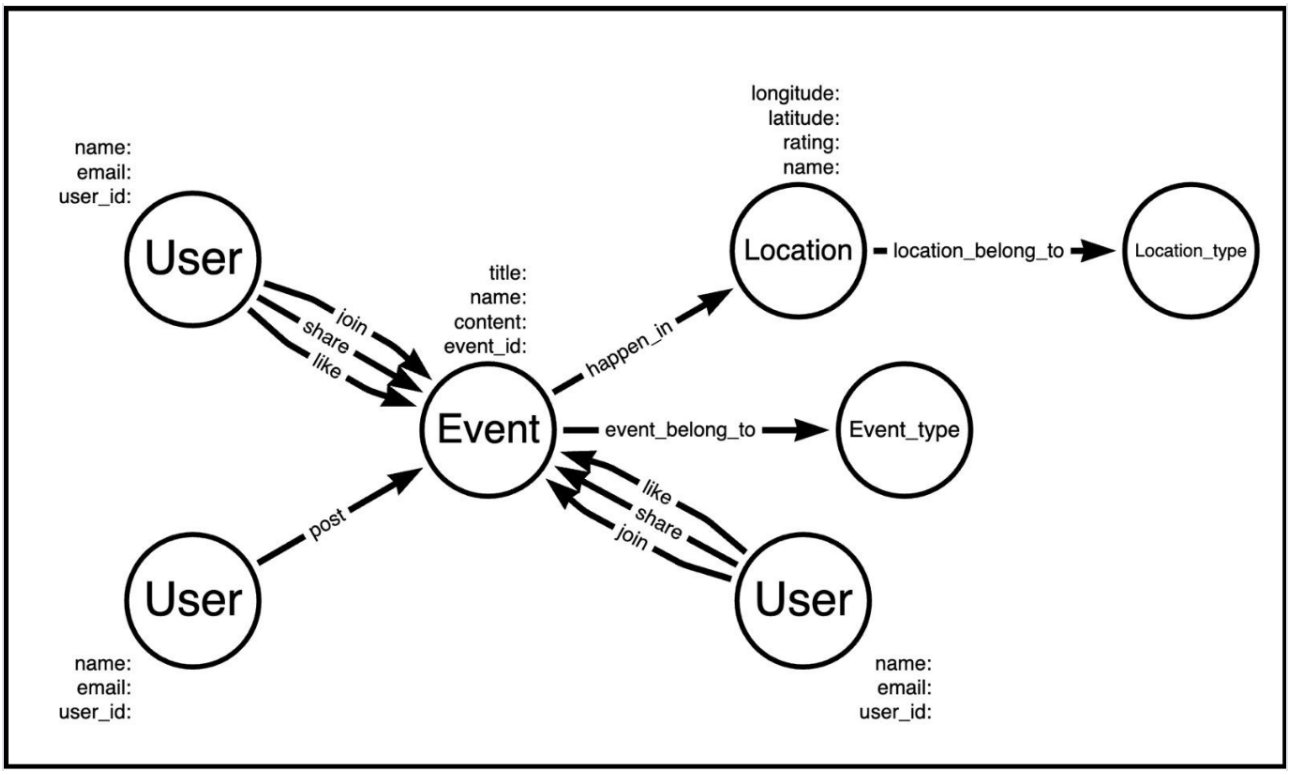

Figure 1: Graph Schema

3.1 Entities and Attributes

- **User**:
  - Attributes: name, email, user_id
- **Event**:
  - Attributes: title, name, content, event_id
- **Location**:

  - Attributes: longitude, latitude, rating, name

- **Location_type**:
  - No specific attributes listed
- **Event_type**:
  - No specific attributes listed

3.2 Relationships

- **User** can *join*, *share*, and *like* an **Event**.
- **Event** *happens in* a **Location**.
- **Event** *belongs to* an **Event_type**.
- **Location** *belongs to* a **Location_type**.
- **User** can *post* an **Event**.

The following Cypher query outlines the creation of the graph schema as described in the design. It includes nodes for **User**, **Event**, **Location**, **Location_type**, and **Event_type**, as well as the relationships between these nodes.

```
// Create User nodes
CREATE (:User {name: 'Alice', email: 'alice@example.com', user_id: '1'}),
       (:User {name: 'Bob', email: 'bob@example.com', user_id: '2'}),
       (:User {name: 'Charlie', email: 'charlie@example.com', user_id: '3'});

// Create Event nodes
CREATE (:Event {title: 'Concert', name: 'Summer Fest', content: 'Music concert',
    ↪ event_id: 'e1'}),
       (:Event {title: 'Conference', name: 'Tech Talks', content: 'Technology
    ↪ conference', event_id: 'e2'});

// Create Location nodes
CREATE (:Location {longitude: 40.7128, latitude: -74.0060, rating: 4.5, name: '
    ↪ Central Park'}),
       (:Location {longitude: 34.0522, latitude: -118.2437, rating: 4.7, name: 'LA
    ↪ Convention Center'});

// Create Location_type nodes
CREATE (:Location_type {name: 'Park'}),
       (:Location_type {name: 'Convention Center'});

// Create Event_type nodes
CREATE (:Event_type {name: 'Music'}),
       (:Event_type {name: 'Technology'});

// Create relationships between Users and Events
MATCH (u1:User {user_id: '1'}), (e1:Event {event_id: 'e1'})
CREATE (u1)-[:POST]->(e1);

MATCH (u2:User {user_id: '2'}), (e2:Event {event_id: 'e2'})
CREATE (u2)-[:POST]->(e2);

MATCH (u3:User {user_id: '3'}), (e1:Event {event_id: 'e1'})
CREATE (u3)-[:JOIN]->(e1),
       (u3)-[:LIKE]->(e1),
       (u3)-[:SHARE]->(e1);
```

```
33
34 // Create relationships between Events and Locations
35 MATCH (e1:Event {event_id: 'e1'}), (l1:Location {name: 'Central Park'})
36 CREATE (e1)-[:HAPPEN_IN]->(l1);
37
38 MATCH (e2:Event {event_id: 'e2'}), (l2:Location {name: 'LA Convention Center'})
39 CREATE (e2)-[:HAPPEN_IN]->(l2);
40
41 // Create relationships between Locations and Location_types
42 MATCH (l1:Location {name: 'Central Park'}), (lt1:Location_type {name: 'Park'})
43 CREATE (l1)-[:BELONG_TO]->(lt1);
44
45 MATCH (l2:Location {name: 'LA Convention Center'}), (lt2:Location_type {name: '
    ↪ Convention Center'})
46 CREATE (l2)-[:BELONG_TO]->(lt2);
47
48 // Create relationships between Events and Event_types
49 MATCH (e1:Event {event_id: 'e1'}), (et1:Event_type {name: 'Music'})
50 CREATE (e1)-[:BELONG_TO]->(et1);
51
52 MATCH (e2:Event {event_id: 'e2'}), (et2:Event_type {name: 'Technology'})
53 CREATE (e2)-[:BELONG_TO]->(et2);
```

Listing 1: Property Graph Schema Definition

This query creates the necessary nodes and relationships to form the graph schema as per the design. The nodes include sample data for users, events, locations, location types, and event types, and the relationships illustrate how these nodes interact with each other.

4 Graph Population Flows (M4)

To populate the graph, it is essential to identify reliable sources of data and design efficient flows to incorporate this data into the graph. The process should be (semi-)automatic to ensure efficiency and accuracy.

4.1 Data Sources

- **User Data**: User data stored in our database.
- **Event Data**: Post data stored in our database.
- **Location Data**: Derived from the post data.
- **Location_type Data**: Generated using a data generation tool such as Faker.
- **Event_type Data**: Sourced from the database.

4.2 Population Process

The population of the graph can be automated using Cypher queries to load data from CSV files. Below are the queries used to populate the graph:

```
1 LOAD CSV WITH HEADERS FROM 'https://2015.filemail.com/api/file/get?filekey=
    ↪ kmwxLI1lFH0UWxLigxXkTERcohSGl3gMPRfjvSMzz4wzF31jhUQ944_L5TCqz05nAGnK&pk_vid=74
    ↪ df05d49e0012fe1717436572beab92' AS line
2 CREATE (:User {
3   id: line.id,
4   name: line.name,
5   email: line.email
```

```
})


LOAD CSV WITH HEADERS FROM 'https://2025.filemail.com/api/file/get?filekey=
    ↪ vYhKGMwFcbDY7aI8x3UxunXFx4E0U8CZlMng0zNHrlbUojUQcnG3V2oy37bGQJFCHZhdNg&pk_vid
    ↪ =74df05d49e0012fe1717439085beab92' AS line
CREATE (:Event {
  id: line._id,
  title: line.title,
  content: line.content
})


LOAD CSV WITH HEADERS FROM 'https://2001-2.filemail.com/api/file/get?filekey=
    ↪ qwiDErqm5_Msjewsq0TF9k7EV0x5qizgtldI2lLEgoFL7BRcMXZgnJGVPy_0-yCtVZa1_4nDXQ' AS
    ↪  line
CREATE (:Location {
  longitude: line.longitude,
  latitude: line.latitude,
  name: line.name
})


LOAD CSV WITH HEADERS FROM 'https://download1654.mediafire.com/94
    ↪ v4andoi3ogLYM5cgy6ZlKnhcFkJgI1R4V
uZwU7gaoszMtVX795zAFbutelqTuRv5NaCuTjLhWDwk4WChx0fLmiEkvcdLJDS1DnQ2FH7mmiNE700a0NcbGbbW5X
    ↪
UNAkYfMppEYSUiwtoTaNMU5iJSx6NlmHqQUDbTOHagqm3A/ziswuhz3kzm4auo/eventtype_semantic.csv
    ↪ ' AS line
CREATE (:EventType {
  id: line.type_id,
  type: line.type
})

LOAD CSV WITH HEADERS FROM 'https://download1501.mediafire.com/h1nztherqdfgIDDn35vKOw
B5dRxb_J7VDVac3DGT7Q6CCh1_df5TiX4ZWQzHsEUeSekdjA2_NksVmfE7Yii0vprAtYi9ekPpHErSNXBt
rQlUCQQAzJq2Lak_6mU--KW3qHOpOX-xd5839sm711ohQjRKsELYEwdf7WVjch4dA/13qc05wy4sg3la
2/user_post_event.csv' AS line
            MATCH (user:User {id: line.user_id})
            WITH user, line
            MATCH (event:Event {id: line.event_id})
            CREATE (user)-[w:post]->(event)


LOAD CSV WITH HEADERS FROM 'https://download1322.mediafire.com/4sgcz7h4d7kgHhwkd
I4OIicQ6NQMS0W4iKRd_heOmCxMMDdOgFEJYTNy91AtXSQHX8ioh5bVXE397GzXj1GsVK2f1Q1
cHLQQbEETF7XvTaSRbYFGPSt52wsX9Ugku010Y3gf3caFWVXO
UxRjbDOrhw2yCbmkSVNT1e-bntrwrQ/iyrmzol4ksxoiaj/user_join_event.csv' AS line
            MATCH (user:User {id: line.userid})
            WITH user, line
            MATCH (event:Event {id: line.eventid})
            CREATE (user)-[w:join]->(event)


LOAD CSV WITH HEADERS FROM 'https://download1654.mediafire.com/94v4an
doi3ogLYM5cgy6ZlKnhcFkJgI1R4VuZwU7gaoszMtVX795zAFbutelqTuRv5NaCuTjLh
WDwk4WChx0fLmiEkvcdLJDS1DnQ2FH7mmiNE700a0NcbGbbW5X
UNAkYfMppEYSUiwtoTaNMU5iJSx6NlmHqQUDbTOHagqm3A/ziswuhz3kzm4auo/
eventtype_semantic.csv' AS line
CREATE (:EventType {
  id: line.type_id,
```

```
  type: line.type
})

LOAD CSV WITH HEADERS FROM 'https://download1322.mediafire.com/
    ↪ m34y56mujyngGPUpEUytLbvbj3Vo7x7sduUHxSSGEwkw04G4Ly
NuN2n1khCB2_Z4YNOOj7Mz2Ym1O7XHLcdW2sEobHO6pG-96nv1-
    ↪ LLxeWhtrhbBXEdETw0BohZamJwmtKiGa2bA2GSayxJbUjg3Y4T3w3mkG9lzk04YL9_Aww/
    ↪ v63ikq53f3qoo5m/user_share_event.csv' AS line
            MATCH (user:User {id: line.userid})
            WITH user, line
            MATCH (event:Event {id: line.eventid})
            CREATE (user)-[w:share]->(event)

LOAD CSV WITH HEADERS FROM 'https://download1509.mediafire.com/
    ↪ l08m3dz4712gHqhIA4HxiUloBiRWEt3MuOCAnk2i96LkLcKw
_sA5wexuVzJawyi0xYFg8fWhaCcNrOyUmkZdqVbrUxStW0aRndYZ_Vz17cAxlien-
    ↪ adNOkv6FWbQidOrSvImdQLiaOMGA1ke-v59WDOtUZmIVCYyr-tt4WhM2Q/slh9h7as04nhcrq/
    ↪ event_belong_to_type.csv' AS line
            MATCH (event:Event {id: line.id})
            WITH event, line
            MATCH (type:EventType {id: line.type_id})
            CREATE (event)-[w:belongto]->(type)

LOAD CSV WITH HEADERS FROM 'https://download1529.mediafire.com/rwom0ypwbjwg_g1dFap7J
8wnQEAPJ_sVCCdllv6OF_66ch8IRk0uWtFr86ittv54grkwwBw1ac5PxgDMEtxQ1_APtD_zrJva12HpfFYp
YWSc9ozsK2claat_D0LtY0i2pg1gIjE4gFpS5NUvZHika0XJfA38ZUW4-KJkjyPVdw/fhu8ii0xdryjgz4/
    ↪ event_happend_in.csv' AS line
            MATCH (event:Event {id: line.event_id})
            WITH event, line
            MATCH (location:Location {name: line.location_name})
            CREATE (event)-[w:happendin]->(location)
```

Listing 2: Graph Population Queries

This set of queries ensures that data from various sources such as user data, event data, location data, event types, and the relationships between them are accurately populated into the graph database. By automating the data ingestion process, we can maintain the accuracy and timeliness of the data in the graph.

5 BPMN Diagram Explanation for Data Integration and Management (M5)

The BPMN diagram illustrates the automated data integration process in the AmiGo app, utilizing a property graph-based catalog for enhanced event recommendations and social interactions, aligning with the goals set out in M1 for accurate event suggestions via sophisticated graph-based analysis.

Data Collection, Ingestion, and Preprocessing

The process begins with **collecting data** from various sources such as user-event participation databases, location systems, event databases, and user information databases. This data is **validated** and stored in a staging area to ensure accuracy before proceeding. The validated data is then **cleaned and preprocessed** to standardize formats, normalize text, and remove duplicates or irrelevant data. Following this, the data is **mapped to a graph schema**, transforming relational data into nodes and edges suitable for a property graph.

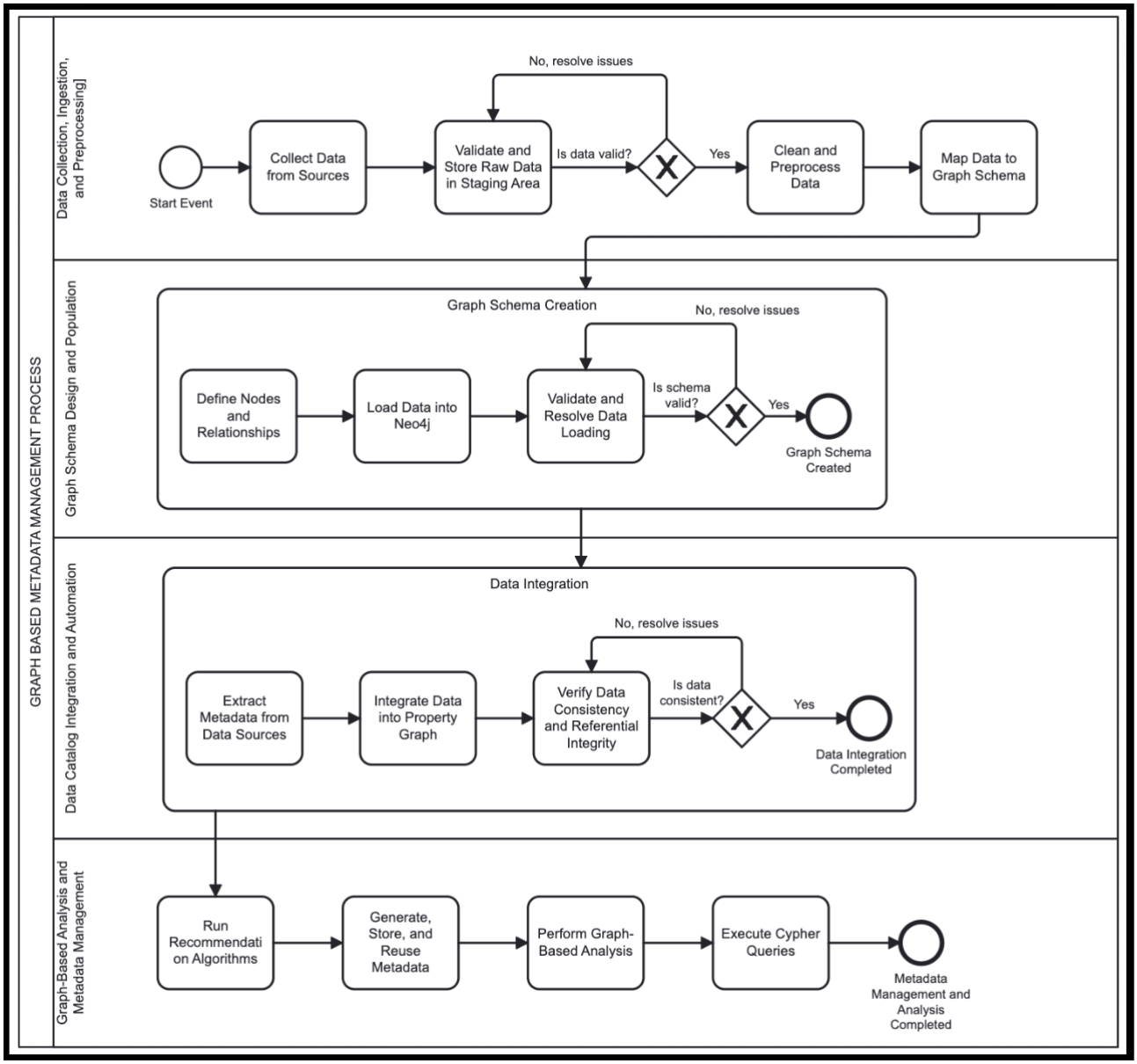

Figure 2: BPMN Diagram for Data Integration and Management in the AmiGo Project

Graph Schema Design and Population

The graph schema is created by **defining nodes and relationships** (e.g., users, events, locations, join, like, share). The mapped data is then **loaded into the Neo4j graph database**, ensuring it is structured according to the schema. The data loading process is **validated** to maintain the integrity and consistency of the graph database.

Data Integration

Metadata, such as user activities and event details, is **extracted** from various data sources and integrated into the property graph. This integration establishes connections between different data points, facilitating complex queries and analyses. **Consistency checks** ensure data integrity across the graph, verifying that relationships between nodes are accurate and necessary references are maintained.

Graph-Based Analysis and Metadata Management

Furthermore, the integrated data is used to run **recommendation algorithms**, leveraging relationships and properties within the graph to provide personalized event recommendations. Metadata generated from analysis and user interactions is **stored for future use**, helping to refine recommendations and improve system performance. **Advanced graph-based analysis** techniques uncover patterns and insights within the data, enhancing user experience and engagement. Finally, **Cypher queries** are executed to retrieve specific information from the graph database, essential for generating actionable insights.

So, our BPMN diagram effectively shows how the automated data integration and management process using a property graph-based catalog within the AmiGo app. By following these steps, the project can handle complex queries efficiently, provide accurate event recommendations, and enhance user engagement through personalized interactions. This process ensures data integrity and consistency while leveraging advanced graph analytics to deliver valuable insights and improve the overall user experience.

6 Metadata Generation and Management - (M6)

Metadata plays a crucial role in maintaining the integrity and usability of our graph-based recommendation system. This section details the processes involved in metadata generation, storage, and reuse.

Metadata Generation

1. **Source Integration:** Metadata is generated during the ETL process, where data from various sources is extracted, transformed, and loaded into the graph. This includes details such as data source, extraction date, and transformation logic.
2. **Relationship Tracking:** As new relationships (edges) are formed in the graph (e.g., user joins an event), metadata capturing the nature and context of these relationships is generated. This includes timestamps, relationship type, and involved entities.
3. **Algorithm Outputs:** The results of recommendation algorithms (e.g., relevance scores, predicted interests) are stored as metadata to enhance future queries and analyses.

   Metadata StorageGraph Schema Extension: Metadata is stored within the graph itself, extending the schema to include metadata nodes and relationships. For example, a metadata node may store the extraction date for a user's interaction. **Efficient Retrieval:** By storing metadata within the graph, it can be efficiently retrieved using Cypher queries, enabling quick access to contextual information during recommendation generation and analysis.

Metadata Reuse

1. **Enhanced Query Performance:** Metadata helps in optimizing queries by providing additional context, such as filtering data based on extraction dates or prioritizing recent interactions.
2. **Improved Data Integration:** Metadata aids in automating data integration tasks. For instance, knowing the source and transformation logic of data helps in seamlessly integrating new data sources.

3. **Support for Learning Algorithms:** Metadata is crucial for machine learning algorithms that require contextual information for training and validation. For instance, metadata about past recommendation success rates can be used to refine the recommendation model.

BPMN Process for Metadata Management

1. Extraction: Data is extracted from sources and metadata is generated capturing source details and extraction context.
2. **Storage:** Metadata is stored in the graph, linked to relevant nodes and relationships.
3. **Utilization:** Metadata is used to enhance query performance, support learning algorithms, and facilitate data integration.
4. **Feedback Loop:** Continuous feedback from system performance and user interactions is integrated as new metadata, ensuring the system evolves and improves over time.

By effectively managing metadata, our project ensures robust data integrity, efficient query performance, and continuous improvement in recommendation accuracy, ultimately enhancing the user experience within the AmiGo app.

7 Proof of Concept (PoC) (M7)

In this section, we implement a proof of concept (PoC) to demonstrate the feasibility and functionality of the graph schema and population flows described in the previous sections. The PoC will utilize Neo4j, a popular graph database, along with Cypher, its query language, to manage and query the graph data. The tools and technologies chosen for this PoC are justified based on their capabilities and suitability for handling graph data.

7.1 Tools and Technologies

- **Graph Database: Neo4j**
  - Neo4j is a robust and scalable graph database that supports ACID transactions and provides high availability.
  - It is highly optimized for handling complex graph queries and supports a rich set of graph algorithms.
  - The use of Neo4j allows for efficient storage and retrieval of graph data, making it an ideal choice for this PoC.
- **Query Language: Cypher**
  - Cypher is a powerful declarative graph query language designed specifically for Neo4j.
  - It provides an intuitive and expressive syntax for creating, reading, updating, and deleting graph data.
  - Cypher's ability to efficiently handle pattern matching and graph traversals makes it suitable for complex queries required in this PoC.

7.2 PoC Setup

1. **Graph Schema Definition**
   - The graph schema is defined using Cypher queries to create nodes and relationships for **User**, **Event**, **Location**, **Location_type**, and **Event_type**.
   - Nodes and relationships are created as described in Section M3, ensuring the graph structure supports the required data interactions.
2. **Data Population**
   - Data is populated into the graph database using Cypher queries to load CSV files.
   - The queries handle the creation of nodes and relationships, ensuring data integrity and consistency.
3. **Execution Example**
   - An example Cypher query is executed to showcase the functionality of the graph database.
   - This includes querying for events posted by a specific user, finding events happening at a particular location, and retrieving users who have joined or liked an event.

7.3 Example Cypher Queries

```
MATCH (e:Event)<-[r:JOIN|LIKE]-(u:User)
WITH e, count(r) AS popularity
ORDER BY popularity DESC
LIMIT 5
WITH collect(e) AS popularEvents
MATCH (recommendedEvent:Event)
WHERE NOT recommendedEvent IN popularEvents
RETURN recommendedEvent.title, recommendedEvent.content
```

Listing 3: Event Recommendation Query

This query finds the most popular events based on the number of ‘JOIN‘ and ‘LIKE‘ relationships from users and recommends other events to users by listing the top 5 popular events along with their details.

8 Evolved Graph Schema Design

We also tried to improve our property graph to make more powerful and effective.

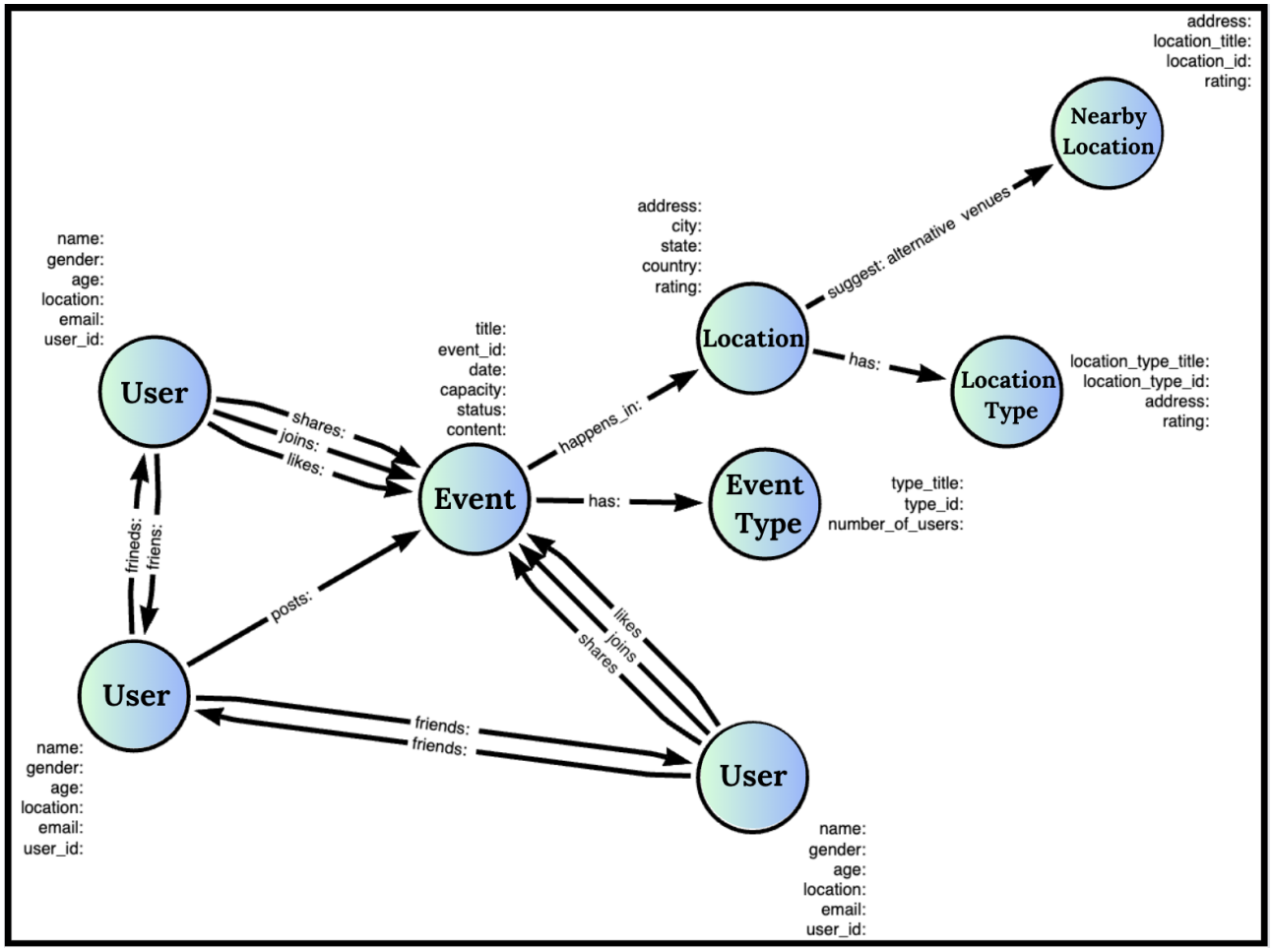

Figure 3: Evolved Graph

Comparison of Initial and Evolved Property Graphs

Summary

- **Nodes**:
  - **User**: Added *gender*, *age*, *location* properties.
  - **Event**: Added *date*, *capacity*, *status* properties.
  - **Location**: Changed properties to include *address*, *city*, *state*, *country*.
  - **Location Type**: Added *location_type_title*, *location_type_id*, *address*, *rating*.
  - **Event Type**: Added *type_title*, *type_id*, *number_of_users*.
  - **Nearby Location**: New node added with properties *address*, *location_title*, *location_id*, *rating*.
- **Relationships**:
  - **User to User**: Introduced *friends* relationship.

Aspect	Initial Property Graph	Evolved Property Graph
Nodes		
User	name, email, user_id	name, gender, age, location, email, user_id
Event	title, name, content, event_id	title, event_id, date, capacity, status, content
Location	longitude, latitude, rating, name	address, city, state, country, rating
Location Type	name	location_type_title, location_type_id, address, rating
Event Type	name	type_title, type_id, number_of_users
Nearby Location	Not present	address, location_title, location_id, rating
Relationships		
User to User	post	friends, friends
User to Event	joins, shares, likes	shares, joins, likes, posts
Event to Location	happens_in	happens_in
Event to Event Type	event_belong_to	has
Location to Location Type	location_belong_to	has
Location to Nearby Location	Not present	suggest_alternative_venues

Table 1: Comparison of Initial and Evolved Property Graphs

– **User to Event**: Added *posts* relationship.

– **Event to Event Type**: Changed relationship name to *has*.

– **Location to Location Type**: Changed relationship name to *has*.

– **Location to Nearby Location**: New relationship *suggest_alternative_venues* added.