Opioid Knowledge Graph

Hayley Baek, Aditi Das, Mousumy Kundu
December 5, 2025
Professor Armanath Gupta
DSE 203 - Data Integration and ETL
University of California San Diego

Project Background

The opioid epidemic began in the 1990s by pharmaceutical companies pushing aggressive advertising campaigns and engaging in misleading marketing tactics to overprescribe opioid painkillers. These plans were discussed through internal emails that were used as pieces of evidence in litigation. Within the US, this also led to a large state response of passing legislation to stifle high-risk prescribing practices. Our goal in this project is to create a knowledge graph from these email documents that links the main actors and events to illustrate compliance in the epidemic and inform public policymakers of new connections within the opioid epidemic.

Data Sources

We integrate multiple data sources in our workflow. First, the SQL collection is hosted by University of California San Francisco and contains the entire database of documents. For our project scope, the collected data is focused strictly on emails from three tables. The emails were ingested into the database as OCR text, so we leverage LLMs as a text index data source to process OCR text into a standardized schema. Many elements of the schema are taken from different object types on Schema.org. Lastly, to add additional robustness into our standardized schema, we also use several APIs as vocabulary sources that are specific to the project's use case.

System Architecture

Our project code environment is hosted on the National Data Platform, and our workspace is launched through the NSF National Research Platform. We're allowed access to this resource specifically for this project as Data Science and Engineering students within University of California San Diego.

The SQL collection is a Postgres and Apache Solr database that has metadata for the documents. While the database contains multiple types of documents like PowerPoint presentations, spreadsheets, and images, our knowledge graph is composed strictly of email messages. Using the `Psycopg` adapter, we establish a connection to the Postgres database and open a server-side cursor to execute SQL queries. The

query results are then fetched into a Python data structure, and the connection is closed. We write a query to select the document IDs across three tables and filter the selection by document type. The selected IDs are then searched in the Solr database using the `urllib` Python package to return texts of entire emails grouped by threads. The email threads exist in Solr as long strings of OCR text and need to be parsed through to become usable for the graph.

We define a standard schema to fit the email data into so that the entities and relationships of the schema's tree-like traversal can directly translate to a graph's path traversal between nodes and edges. We use object types and object properties from Schema.org to define graph node types and attributes, and the relationships between additional levels of data tree are used as graph edge types and attributes. The schema contains properties that are standard for an email message, such as subject line and email body, and for people, such as name and email address. For the use-case of our project, our schema also defines semantically derived attributes. From within the email body text we extract entity types like mentions of locations, finances and events and entity attributes like compliance context and decisions of the emails. To give a high-level overview of the schema, the top layer of the tree-like structure has the legal case. The tree layer below the legal case has emails, as emails are pieces of evidence presented in litigation. Emails are the most attribute-rich node type, as all of the semantically derived entities are on the email level. The mentioned entities exist as multiple node types (e.g., *locations, drug names, businesses*) but all exist one layer below emails with the same edge type (*mentions*). Emails are sent and received by people who belong to organizations, so people and organizations are two more node types connected by two more edge types.

Several LLMs are leveraged in the data processing step. The first LLM is the GPT-5 nano model, as it is OpenAI's most cost-efficient model for processing at $0.025 for every million input batch tokens. We use this model to uncouple individual emails from their email threads and fit the data into our schema. The instructions for the model are explicitly to not hallucinate or make inferences from the input text during the schema fitting process and to populate the fields as direct substrings. This is crucial as we want to minimize data loss while restructuring the data through the LLM which can often generate its own ideas that weren't present in the original data. The schema is fed into the model's instructions as a JSON object. The email threads are fed as raw string inputs into the LLM and output in a JSONL file of the emails unnested from their threads where each file line contains a schema.

To begin our semantic entity extraction from the processed data, we introduce two specialized vocabulary libraries: the `spaCy` library to extract chemical and biomedical terms to be matched against `RxNorm`'s `RxCUI` identifier library to get valid, standardized drug names that are mentioned in the emails. Occasionally an email will mention references to another email from a different thread. To link those emails together, we apply a TF-IDF vectorizer and a cosine distance function to score text

similarities between different emails. We define a similarity threshold of 0.25 to determine that two emails are similar enough to each other that they should be considered cross-referential of each other. We also use the Qwen2.5 API as a semantic entity extraction tool to read through the email bodies and add more named entities to the schema like location and event mentions and concerns raised. Qwen3 is able to make these semantic entity additions for about $0.01 for every million input batch tokens. For both GPT-5 nano and Qwen2.5, we process our input data into batches for runtime and cost efficiency, as without batch processing these LLMs need to be run overnight and at over double the cost.

With our fully processed data, we write Cypher queries to create the knowledge graph. We first declare node and edge uniqueness constraints so that duplicates are never created. Then, in creating the nodes and edges, we write upsert helper functions to match and merge the types and attributes of the graph's nodes and edges as they appear in the defined schema. The node types are denoted in the schema as `@type` and edge types are denoted in the schema as the relationship names between the tree layers. The node attributes are the other fields that appear in the same schema level as the `@type` field. We iterate through each line in JSONL output file and apply the upsert functions to populate the graph from the values that are present in the schema.

The raw text contains many spelling inconsistencies due to being OCR text even as they appear in the schema. We resolve this using entity resolution, a process which normalizes the text and leverages the `SequenceMatcher` class from Python's `difflib` module. Incoming nodes are checked against the list of nodes that already appear in the graph using a similarity score to determine whether the two nodes are similar enough to each other, with a particular threshold of certainty, to be merged into one node. If two entities are only slightly different in string differences like case sensitivity or misspellings, then their similarity score will be high, and a redundant node won't be created in the graph. If two nodes have low similarity scores, then they can be determined to be meaningfully different, and the new incoming node will be added to the graph.

Through this workflow we convert around 650 documents to a knowledge graph with around 16,500 nodes and around 31,000 edges.

Data Extraction from SQL Collection

The data is collected from a Postgres database. An engine is created and a connection is established with the database with the specified database host, name, username, and password given to the project group by UCSD. The database tables contain metadata for the documents, and the IDs of those documents are extracted by looping through the tables in a PLSQL query as follows:

```
DO $$
DECLARE
```

```
        table_name TEXT;
        table_array TEXT[] := ARRAY[{table_array_str}];
    BEGIN
        CREATE TEMP TABLE IF NOT EXISTS temp_all_ids (id TEXT) ON COMMIT
    PRESERVE ROWS;
        TRUNCATE temp_all_ids;
        FOREACH table_name IN ARRAY table_array LOOP
            EXECUTE format('INSERT INTO temp_all_ids SELECT id FROM %s WHERE
    lower(type) = ''email''', table_name);
        END LOOP;
    END $$;
```

The variable `table_array_str` holds the table name from a list of table names that the query selects the document ID from. The IDs are then stored in a list to iterate through for the document text data collection.

The document IDs are passed into an f-string with the Solr URL to collect the raw document text. The Python package `urllib` and its function `urlopen` is used to collect the response of the open URL connection. The response is then extracted as a string from the json object and put into a dictionary with the email ID as a key-value pair of the email ID and the document text. The dictionary object is then put into a list so that the items in the list can be iterated through for the LLM.

Document Schema Definition

The raw document text extracted from the SQL collection is processed through a schema. The schema identifies the entities present in the data like emails, people, and locations, and the entities have attributes as outlined by Schema.org. The relationships between the entities are expressed in the tree-like structure of the schema so that the layer traversal in the tree-like structure translates to path traversal in the knowledge graph. The schema is defined as follows:

```
{
  "@context": {
    "@vocab": "https://schema.org/",
    "email": "https://schema.org/EmailMessage",
    "person": "https://schema.org/Person",
    "org": "https://schema.org/Organization",
    "document": "https://schema.org/DigitalDocument",
    "topicEntity": "https://schema.org/Thing",
    "url": "https://schema.org/URL",
    "gpe": "https://schema.org/Place",
```

```
  "drug": "https://schema.org/Drug"
},

# Thread-level wrapper
"@type": "case:Legislation",
"semantic_type": "Legal Communication Record",
"identifier": "",
"legalStatus": "",
"dateFiled": "",
"language": [],   # could be "en" for English or a different language
"confidentialityNotice": "",
# The thread: multiple messages in reverse-chronological order
"hasPart": [
  {
    "@type": "email:EmailMessage",
    "semantic_type": "Email Communication",
    "identifier": "",
    "subject": "",      # substring of raw_thread_text (if present)
    "dateSent": "",
    "importance": "",
    "threadIndex": 0,   # 0 = most recent message
    "inReplyTo": "",    # optional

    "sender": {
      "@type": "person:Person",
      "semantic_type": "Person",
      "name": "",       # substring of raw_thread_text
      "email": "",      # substring of raw_thread_text
      "affiliation": {
        "@type": "org:Organization",
        "semantic_type": "ORG",
        "name": "",     # substring if present, else empty
        "role": "",

        "parentOrganization": {
          "@type": "org:Organization",
          "semantic_type": "ORG",
          "name": "",
```

```
      "role": ""
    }
  }
},

"recipient": [
  {
    "@type": "person:Person",
    "semantic_type": "Person",
    "name": "",     # substring
    "email": "",    # substring if present
    "affiliation": {
      "@type": "org:Organization",
      "semantic_type": "ORG",
      "name": "",
      "role": ""
    }
  }
],

"body": "", # MUST be a literal substring of raw_thread_text

"mentions": [  # MENTIONS: text grounded, labels inferred
  {
    "@type": "topicEntity",
    "semantic_type": "", # e.g. "Financial Document", "Drug Name", "GPE"
    "role": "",            # e.g. "Geographic Destination"
    "name": "",  # MUST be substring of the user input text
    # MUST be substring if it appears in the text; otherwise leave ""
    "identifier": ""
  }
],

# ATTACHMENTS: names/desc grounded, format may be inferred
"attachments": [
  {
    "@type": "document:DigitalDocument",
    "semantic_type": "",   # e.g. "Spreadsheet Document", "Presentation”
```

```
            # MUST be substring of raw_thread_text
            "name": "",            # e.g. "TLAIS costings 8.09.04.xls"
            "fileFormat": "", # may be inferred (application/pdf, etc.)
            # MUST be substring if the description text appears in the email;
            # otherwise either leave "" or omit the field.
            "description": ""
          }
        ],

        "forwardedMessage": None,
        "mentionsEmail": [],
        "structuredArgument": [],
        "complianceContext": {
            "@type": "CreativeWork",
            "semantic_type": "",
            "name": "",
            "keywords": [],
            "about": []
        }
      }
    ]
}
```

Text Processing into Schema

After collecting the raw text data and defining the schema to reformat the data into, the text and schema are both passed through the GPT-5-nano model using the OpenAI Batch API for runtime and cost efficiency. The GPT-5-nano is chosen over GPT-4o-mini because it's the most cost-efficient at $0.025 per million batch input tokens. Because of GPT-5-nano's higher tendency to make interpretations, more strict system instructions are outlined so that the model only returns the identified entities in the raw text as substrings with minimal hallucinations or interpretations. The system instructions are as follows:

```
- The user input is a single email thread (one CSV row).
- Split the thread into individual email messages.
- For each message, create one object in `hasPart` with @type
email:EmailMessage".
- Order `hasPart` in reverse-chronological order so index 0 is the most
recent message.
GROUNDING RULES (VERY IMPORTANT):
```

- `body` MUST be a literal substring of the user input text. Do not paraphrase or summarize.
- `subject`, sender/recipient `name` and `email`, attachment `name`, and any description text MUST be literal substrings of the user input text where they appear.
- For each `mentions` item:
  - `name` MUST be a literal substring of the user input text.
  - `identifier` MUST be a literal substring of the user input text if present; otherwise leave "".
  - `semantic_type` and `role` are categorical labels and MAY be inferred.
- For each `attachments` item:
  - `name` MUST be a literal substring of the user input text.
  - `description` MUST be a literal substring if such text exists; otherwise use "" or omit the field.
  - `fileFormat` MAY be inferred (e.g., "application/vnd.ms-excel" for .xls) and does not need to match any literal span.
- Do NOT introduce any text in `body`, `name`, `description`, `subject`, or other free-text fields that is not a substring of the user input text.
- You may assign or infer categorical labels such as:
  - mentions.semantic_type: "Legal Case", "Product Brand", "Business Operation",
    "Financial Document", "Drug Name", "GPE", etc.
  - mentions.role: e.g., "Geographic Destination".
  - attachments.semantic_type: "Spreadsheet Document", "Presentation Document",
    "Policy Document", etc.
  - attachments.fileFormat: e.g., "application/vnd.ms-excel", "application/pdf", etc.
  - complianceContext.semantic_type: e.g., "Regulatory and Legal Framework".
  - complianceContext.name: e.g., "Export Control and Restricted Destination Compliance".
- Do NOT summarize the thread; your job is to structure it, not rewrite it.
- Output strictly must be valid JSON only (no markdown or commentary).
- Follow this JSON structure exactly: {schema}

Since the GPT-5-nano model has a context window of around 400k input tokens and a maximum output of 128k tokens, the batches need to be split into smaller chunks before passing the documents through the model. Since the emails dataset is 652 items long and the longest document is identified to be around 400k characters long, each batch chunk is set to hold 300 items. For very long document texts, this chunk will need to be set to a smaller limit to accommodate the limit of input tokens per batch. For emails with no bodies, the LLM should return a schema with an empty string for the email body entity. Functions for creating the batch, submitting the batch, displaying the status of the batch, and collecting the results of the batch are defined in the code. The function calls for the entire batch pipeline take around $0.75 of OpenAI credits and around 30 minutes to fully process the raw email text into a JSONL file where each line of the file is a schema that is a cleaned version of the input text.

Additional Entity Extraction

(i) Adding cross reference email Ids

Tracking emails that belong to the same discussion thread is challenging when subject lines vary, replies lose metadata, or context is fragmented across multiple chains. Emails may contain highly similar content while appearing under different threads, leading to a loss of continuity. To reconstruct these broken conversations, we introduce a cross-referencing mechanism based on TF–IDF vectorization and cosine similarity. Each email body is extracted and converted into a TF–IDF matrix using the ‘sklearn’ library, producing a numerical representation that captures the semantic fingerprint of the message. Before adding tfidf we applied english stopwords to get the filtered text. Cosine similarity is then computed between email vectors to measure how closely related two texts are. Emails that exceed a similarity threshold of 0.25 are treated as cross-referential, indicating that they likely belong to the same conceptual conversation even if their metadata does not explicitly link them. This approach enables the restructuring of fragmented threads and the discovery of implicit conversational relationships across the dataset.

An example of cross-reference section added in JSON file:


```
"hasPart": [...],
"crossRefInfo": {
  "crossRefEmails": [
    {
      "cid": "klwf0232",
      "score": 0.6117
    },
    {
      "cid": "tplw0232",
      "score": 0.3039
    },
    {
      "cid": "nznw0232",
      "score": 0.2922
```

```
        },
      ],
      "totalCrossRefs": 3
    }]
```


(ii) Drugs names extraction using RxNorm vocabulary

To accurately identify pharmaceutical substances referenced within the email corpus, we implemented a multi-stage drug name extraction pipeline grounded in the RxNorm standardized vocabulary. The process began by filtering textual terms to remove noise and ensure linguistic validity: tokens shorter than three characters, titles, special characters, and other non-informative elements were excluded. The remaining candidate terms were then processed using spaCy's biomedical vocabulary, which provides domain-specific language models capable of recognizing medically relevant entities. Finally, these filtered and spaCy-validated terms were matched against the RxNorm RXCUI vocabulary to resolve them to canonical drug identifiers. This ensured that extracted drug mentions were both semantically meaningful and standardized across the dataset, enabling consistent downstream analysis. An example of drugs RxNorm section in JSON:


```
{
  "email_id": "pxwf0232",
  "output": {
    "@context": {
      "@vocab": "https://schema.org/",
      "email": "https://schema.org/EmailMessage",
      "person": "https://schema.org/Person",
      "org": "https://schema.org/Organization",
      "document": "https://schema.org/DigitalDocument",
      "topicEntity": "https://schema.org/Thing",
      "url": "https://schema.org/URL",
      "gpe": "https://schema.org/Place"
    },
    "@type": "case:Legislation",
    "semantic_type": "Legal Communication Record",
    "identifier": "Case-1-17-md-02804-DAP-Exhibit-651-Emails",
    "legalStatus": "Confidential document disclosed in litigation",
    "dateFiled": "2007-09-27",
    "language": [
      "en"
    ],
    "confidentialityNotice": "CONFIDENTIAL - SUBJECT TO PROTECTIVE
ORDER",
    "hasPart": [...],
    "crossRefInfo": {
      "crossRefEmails": [...],
      "totalCrossRefs": 7
```

```
    },
    "drugsRXnorm": [
      "oxycodone"
    ]
  }
}
```


(iii) Entity extraction using Qwen2.5 Instruct 7b model

To enrich the email dataset with higher-level contextual meaning, we applied the Qwen 2.5 7B Instruct model to extract semantic entities. We provided Qwen with context and prompt to extract semantic entities like decisions, concerns, events, location, people who were mentioned other than senders and recipients in the conversation and any financial references-elements that are not easily captured through rule-based NLP. This process was run on 650 email objects spanning approximately 2,000 individual messages. The model processes 1,000 API requests or 1 million tokens per day, with a total cost of only $0.30 USD to date. The system instruction is as follows:

```
Analyze the following email body text and extract structured
information.{context_str}
    Email Body:
    {body_text}
    Extract and return a JSON object with the following fields:
    1. "decisions_made": Array of decisions or conclusions
    2. "concerns_raised": Array of concerns, risks, or issues mentioned
    3. "people_mentioned": Array of people mentioned (beyond
sender/recipient)
    4. "locations_mentioned": Array of geographic locations mentioned
    5. "events_mentioned": Array of events mentioned
    6. "financial_mentions": Any financial figures, costs, or budget
items mentioned
    Return ONLY the JSON object, no additional text or markdown
formatting.
```

To execute the LLM efficiently, the emails were processed in batches of ten items per batch, with each batch taking approximately 2.5 to 3 minutes to complete. Any errors encountered during batch execution were logged. Thereafter, all failed batches were reprocessed to ensure full coverage of the dataset. The resulting semantic annotations significantly enhance the interpretability of the email, enabling the knowledge graph to represent not just the content of communications but the underlying intent, risks, and operational actions reflected in the emails. An example of the extracted semantic entities in JSON is as follows


```
"enriched_content": {
        "decisions_made": ["Rite-Aid and Anda shall identify products
that Rite-Aid shall purchase exclusively from Anda"],
```

```
        "concerns_raised": [],
        "people_mentioned": [
              "Denny Murray",
              "Denman Murray"
        ],
        "locations_mentioned": ["Deerfield, IL 60015"],
        "events_mentioned": [],
        "financial_mentions": []
}
```


Entity resolution

Entity Resolution (ER) is the task of detecting when multiple records actually refer to the same person, organization, or object. In real datasets, the same entity may appear with different spellings, partial information, or inconsistent formats. ER compares attributes such as names, emails, and organizations, calculates similarity scores, and determines whether two records should be matched or kept separate. This process helps maintain a clean and accurate knowledge graph by preventing duplicate nodes and consolidating information about the same entity.

Entity Resolution is essential because it ensures that each real-world entity is represented only once in the graph. Without ER, the dataset would contain duplicate nodes caused by spelling variations, incomplete information, or inconsistent formatting. These duplicates reduce data quality, create incorrect or fragmented relationships, and lead to inaccurate query results. By identifying and merging records that refer to the same entity, ER maintains a clean, meaningful, and reliable knowledge graph that supports accurate analysis.

Entity Resolution Example

Entity resolution is the process of identifying and merging multiple records that refer to the same real-world entity. This example demonstrates how entity resolution improves data quality by eliminating duplicates and inconsistencies.

(i) Before Entity Resolution:
Two separate records, person1 and person2, represent the same individual but contain slight variations:

- Name: Burt Rosen vs. Burt E. Rosen
- Email: Both share the same email address burt.rosen@pharma.com
- Organization: Both list Walmart

Due to these differences, the system initially treats them as distinct entities.

(ii) After Entity Resolution:
The system detects that these records refer to the same person and merges them into a single, unified entity:

- Name: Standardized as Burt Rosen
- Email: burt.rosen@pharma.com
- Organization: Walmart

This eliminates duplicate records, improves data accuracy and consistency and enables better analytics and decision-making

Entity Resolution Architecture

This diagram represents the Entity Resolution Architecture, which outlines the steps for identifying and merging duplicate or related records.

(i) Load JSONL Data into Neo4j

- The process begins by loading data in JSON Lines (JSONL) format into Neo4j, a graph database.
- Neo4j is used because entity resolution often involves relationships between entities, which graph structures handle efficiently.

(ii) Candidate Generation (Blocking Strategy)

- To reduce computational complexity, a blocking strategy is applied.
- This step groups similar entities into blocks based on shared attributes (e.g., same email domain, similar names).
- Instead of comparing every record with every other record, comparisons are limited to entities within the same block.

(iii) Similarity Computation

- Within each block, entity pairs are compared using similarity metrics (e.g., Levenshtein distance for names, exact match for emails).
- A similarity score is calculated for each pair, indicating how likely they represent the same entity.

(iv) Classification

- Based on similarity scores and predefined threshold rules, pairs are classified as:
    - Match (same entity)
    - Non-match (different entities)
- This step uses decision logic or machine learning models to make the classification.

(v) Merging

- All duplicates identified as matches are grouped together.
- These are merged into canonical records, creating a single, clean representation of each entity.

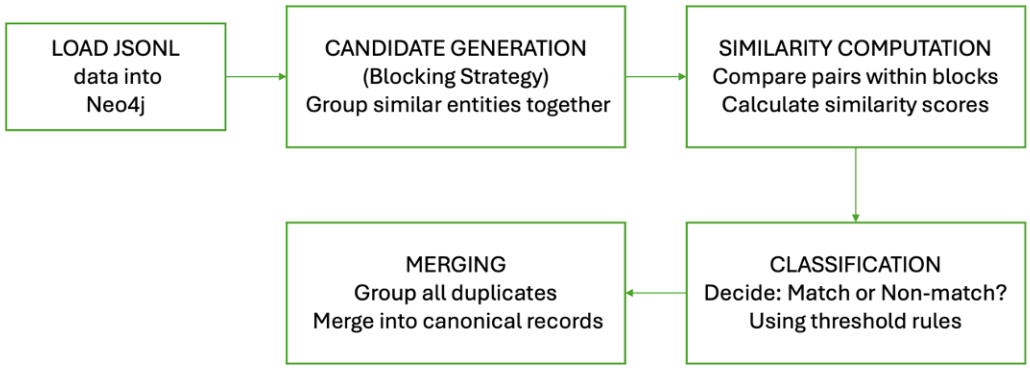

Figure 1: Entity Resolution Architecture

Entity Resolution Rules

To accurately detect and merge duplicate records for the **Person** node, the following rules and strategies are applied:

1. Grouping by Email Domain
   - Persons are grouped based on their email domain.
   - If two persons share the exact same email address, they are considered definite duplicates.

2. Name Similarity (Using SequenceMatcher)
   - Normalize names by:
     - Converting to lowercase
     - Removing punctuation
     - Removing common suffixes (e.g., Inc, LLC, Corp)
   - Compare normalized names using a string similarity ratio.
   - Threshold-based merging:
     - For duplicates in email blocks, use a threshold of 0.9 (default).
     - For duplicates in name-only blocks, use a stricter threshold: 0.9 + 0.05, max 0.98.

3. Blocking by First Word of Normalized Name (Efficiency)
   - Only compare persons who share the first word of their normalized name.
   - Skip very short names (less than 3 letters) and unknown names (e.g., “unknown”, “[unknown]”).

4. Transfer Relationships
   - Maintain relationships when merging:
     - SENT (person → email)
     - SENT_TO (email → person)
     - AFFILIATED_WITH (person → organization)

5. Merge Properties and Delete Merged Node
   - After transferring relationships, merge properties into the canonical record and delete the duplicate node.

Entity Resolution Rules for Organization Node

To detect and merge duplicate records for the Organization node, the following rules are applied:

1. Normalize Organization Names
   - Convert names to lowercase.
   - Remove punctuation and extra spaces.
   - Eliminate common company suffixes (e.g., Inc, LLC, Corp).

2. Compare Normalized Names Using String Similarity

- Use SequenceMatcher to compute similarity between normalized names.
- Apply a default threshold of 0.85 for determining duplicates.

3. Transfer Relationships
   - Preserve relationships when merging:
     - AFFILIATED_WITH (people → organization)
     - SUBSIDIARY_OF (both incoming and outgoing links)

4. Merge Properties
   - Retain important attributes such as semantic_type and role if available in either node.

5. Delete Merged Node
   - After transferring relationships and merging properties, remove the duplicate node to maintain a clean graph structure.

Entity Resolution output

1. Before vs After Entity Resolution (Left Panel)

- **Overall Changes:**
  - Nodes: Reduced from 17,614 to 16,986 (a reduction of 628 nodes, ~3.6%).
  - Relationships: Reduced from 33,898 to 33,432.
- **Node Changes by Type:**
  - Organization: Reduced from 289 to 277 (12 duplicates removed).
  - Person: Reduced from 4,651 to 4,035 (616 duplicates removed).
- **Duplicate Reduction:**
  - Persons: 13.2% of person nodes were duplicates.
  - Organizations: 4.2% of organization nodes were duplicates.
- **Performance:** The entire entity resolution process completed in 12.98 seconds.

2. High-Similarity Matches (Top Right Panel)
- Shows examples of pairs with similarity score = 1.000, meaning they are exact matches:
  - Example 1: Erik Lilje vs Erik Lilje (same email).
  - Example 2: Connie Woodburn vs Connie R. Woodburn (same email).

3. Organization Name Merging (Bottom Right Panel)
- Displays examples of organization name merges with similarity scores:
  - walgreens → walgreen (similarity: 0.84)
  - teva pharmaceuticals → teva pharmaceutical (similarity: 0.97)
  - qualitest → qualitest (similarity: 0.98)
- These merges are based on normalized names and similarity thresholds.

Here, the process successfully removed duplicates and merged entities, improving data quality. Most duplicates were in Person nodes, indicating frequent variations in names and emails. Organization duplicates were fewer but still significant. The pipeline is efficient, completing in under 13 seconds.

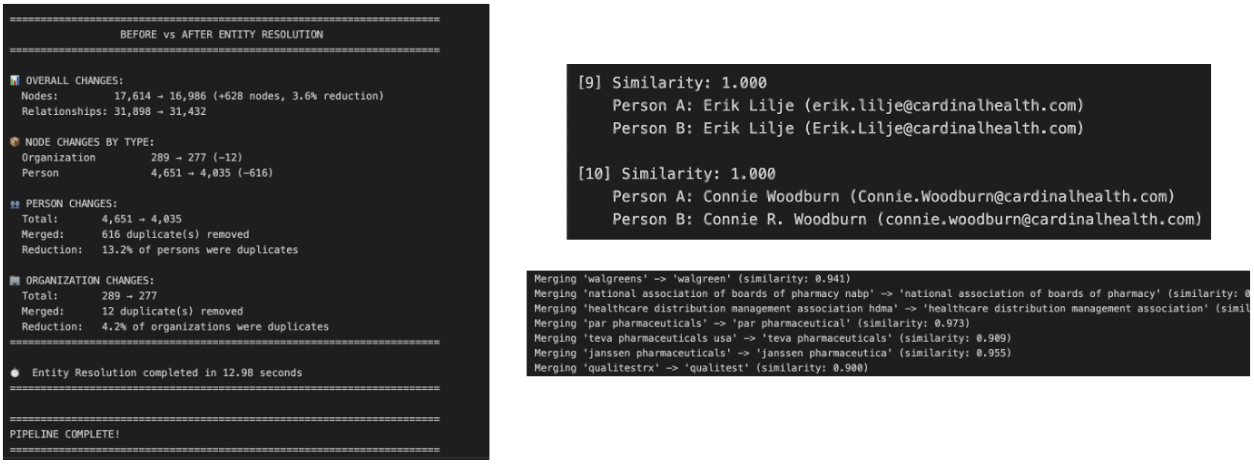

Figure 2: Entity Resolution Output screenshots

Knowledge Graph

Graph Enrichment: Creating Additional Drug-Related Relationships

After the initial graph construction, the underlying dataset contained only the following path for drug-related information:

Person → SENT → Email → EMAIL_MENTIONS_DRUG → RxNormDrug

While this structure preserves the original source relationships, it does not provide an efficient way to analyze drug-centric interactions.
To support richer queries and downstream analytics, we generated **new, derived relationships**.

1. Creating New Drug-Oriented Relationships

Based on co-occurrence patterns in the raw data, we introduced several new edges to simplify queries and reveal hidden patterns.

(a) Person → DISCUSSES_DRUG → RxNormDrug

A direct edge connecting a person to the drug(s) mentioned in their emails.

Purpose:

- Enables quick identification of people associated with each drug
- Reduces the need to traverse through emails

Rule:
If a person sends an email that mentions Drug X, create:
(p:Person)-[:DISCUSSES_DRUG]->(d:RxNormDrug)

(b) RxNormDrug → CO_MENTIONED_WITH → RxNormDrug

A symmetric relationship between pairs of drugs that appear together in the same email.

Purpose:

- Allows analysis of drug co-mention patterns
- Useful for discovering therapeutic combinations or shared contexts

Rule:
If Drug A and Drug B appear in the same email, create:
(A)-[:CO_MENTIONED_WITH]->(B)

(c) Organization → RESEARCHES_DRUG → RxNormDrug

Links organizations to drugs mentioned in emails associated with that organization.

Purpose:

- Shows company-level involvement with specific drugs
- Enables organization-drug trend analysis

Rule:
If a Person belonging to Organization O discusses Drug X, create:
(o:Organization)-[:RESEARCHES_DRUG]->(d:RxNormDrug)

(d) Person → COLLABORATES_ON_DRUG → Person

A derived connection showing collaboration between two people who discuss the same drug.

Purpose:

- Useful for collaboration network analysis
- Helpful for identifying expert communities around specific drugs

Rule:
If Person A and Person B both discuss Drug X, create:
(A)-[:COLLABORATES_ON_DRUG {drug: X}]->(B)

2. Example: Person–Drug Direct Relationship

Creating the DISCUSSES_DRUG Relationship

We generate a direct link from the person to the drug mentioned in the email using Cypher:

```
MATCH (p:Person)-[:SENT]->(:Email)-[:EMAIL_MENTIONS_DRUG]->(d:RxNormDrug)
MERGE (p)-[:DISCUSSES_DRUG]->(d);
```

This ensures that each person has an explicit, query-efficient relationship to every drug they discuss.

3. Generating Direct Drug Relationships

To simplify analysis of drug-related communication, we enriched the graph with two derived relationships: **DISCUSSES_DRUG** and **RECEIVES_DRUG_INFO**.

(a) DISCUSSES_DRUG (Person → Drug)

We first identified all emails in which a person mentioned a drug. For each person–drug pair, we counted the number of such emails and created a direct relationship:

- **Person → DISCUSSES_DRUG → Drug**
- frequency property stores how many times the drug was mentioned
- created stores a timestamp of relationship creation

This transformation collapses a multi-hop pattern
Person → Email → Drug
into a single hop, enabling faster and simpler queries.

(b) RECEIVES_DRUG_INFO (Person → Drug)

We similarly captured the drugs that individuals received information about by detecting all incoming emails that mention drugs.
A direct edge was created:

- **Person → RECEIVES_DRUG_INFO → Drug**
- frequency indicates how many such emails they received
- created indicates when this derived relationship was created

These derived relationships significantly reduce query complexity and allow efficient analysis of communication flows involving pharmaceutical entities.

Case with Most Emails and Associated Persons

The analysis identified Case-17-md-02804-DAP as the case with the highest email activity, totaling 113 emails, significantly more than other cases (e.g., 29 and 15 emails for the next highest cases). Within this case, several individuals exhibited high communication activity. For example, Burt Rosen had 78 sent

emails, 62 received emails, and a total activity of 140, while others like Jack Crowley and Anita Ducca also showed substantial engagement. These findings highlight that this case is communication-intensive, involving multiple key actors from major organizations, which may indicate its complexity and importance.

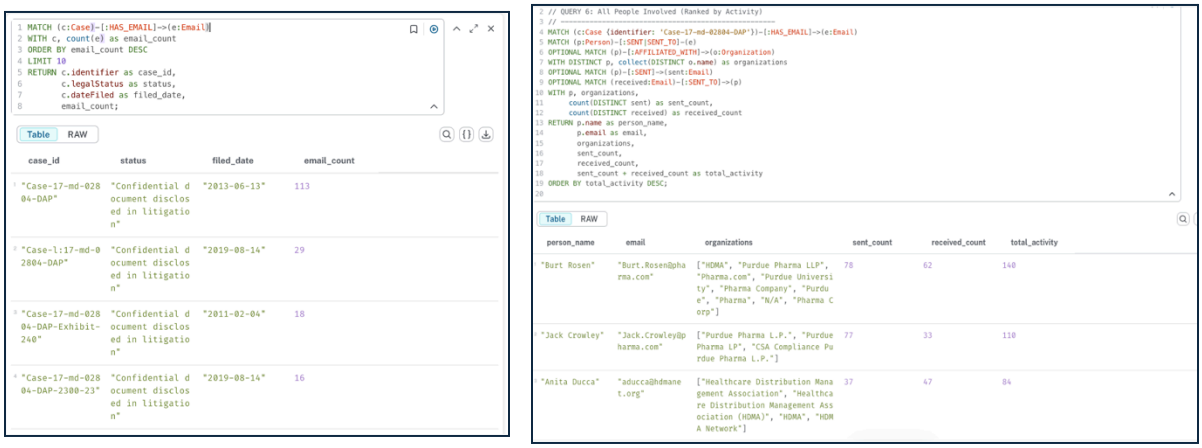

Figure 3: Case id with highest number of emails and the persons are associated with the emails

Visualizing Connections for Case ID: Case-17-md-02804-DAP

This figure 4 illustrates the structure of the knowledge graph built from the dataset. It contains 675 nodes representing different entity types such as emails, cases, decisions, events, locations, persons, and drugs, along with 992 relationships connecting them. The color-coded nodes indicate entity categories, while the edges represent relationships like EMAIL_MENTIONS_TOPIC, SENT_TO, and HAS_DECISION. This visualization highlights how emails serve as central hubs linking various entities, revealing patterns of communication and topic associations within the dataset.
Finds all nodes and relationships within 1 to 3 hops of the case. This is used to generate a graph visualization of the case context—showing related people, organizations, emails, documents, drugs, etc.

//cypher query

```
MATCH path = (c:Case {identifier: 'Case-17-md-02804-DAP'})-[*1..3]->(n)
RETURN path
LIMIT 1000;
```

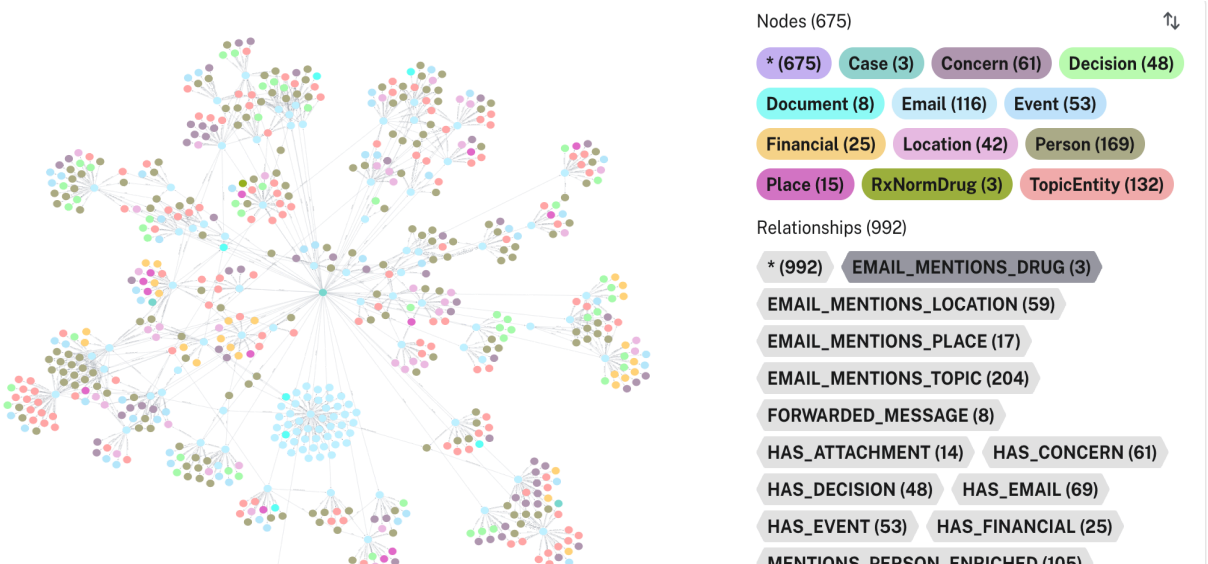

Figure 4: Visualizing Case-Level Neighborhood (Nodes + Relationships) for Case ID: Case-17-md-02804-DAP

Drug–Person–Organization Relationships for Case-17-md-02804-DAP

The left panel shows a graph visualization of entities and their relationships extracted from the Opioid Knowledge Graph. Each node represents an entity such as Person, Case, or Organization, and edges represent the connections between these entities. The color coding indicates different entity types:

- Green: Organization
- Purple: Person
- Brown: Case

This visualization provides an overview of how cases, individuals, and organizations are interconnected within the dataset.

The right panel displays the query results from Neo4j. The Cypher query retrieves drug names and their corresponding mention counts from the knowledge graph. The table lists the top drugs mentioned in the dataset:

- coco diethanolamide: 1950 mentions
- hydrocodone: 556 mentions
- dobutamine: 508 mentions
- oxycodone: 470 mentions
- oxycontin: 281 mentions

This analysis highlights the most frequently mentioned opioids, which can be useful for identifying trends and prioritizing further investigation.

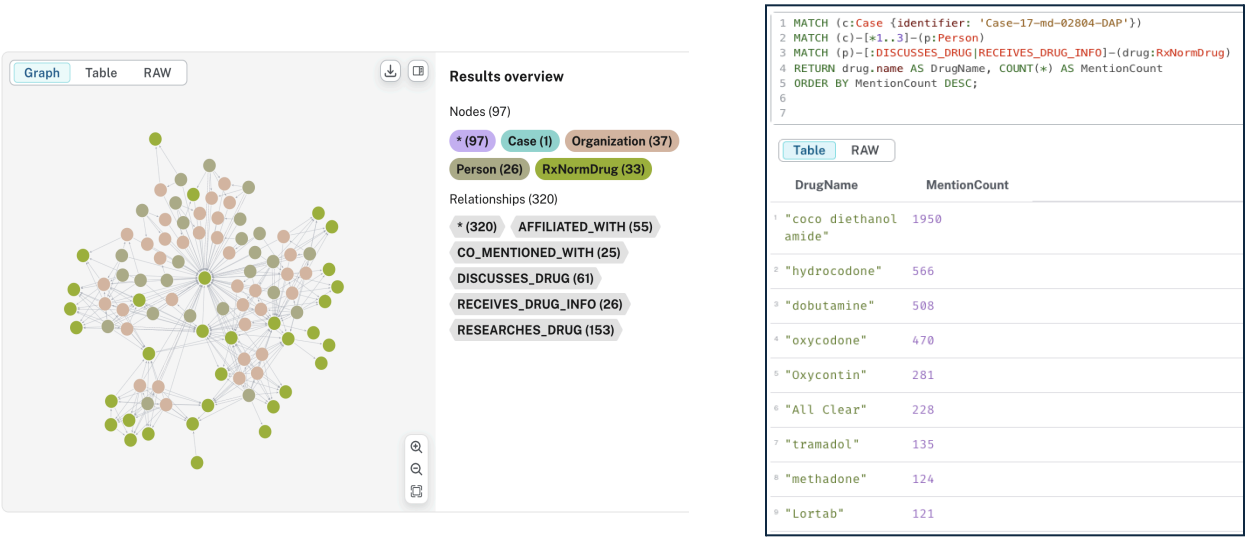

Figure 5: Drug–Person–Organization Network for Case-17-md-02804-DAP. Left: Network graph showing relationships among entities (Person, Case, Organization). Right: Neo4j query output listing opioid drug names and their mention counts, with *coco diethanolamide* being the most frequent.

Instructions to run the code

The code is available on GitHub:

https://github.com/aditidas3/ETL_Opioid_Knowledge_Graph/tree/main.

In the Opioid Knowledge Graph_Part3.ipynb file within the main pipeline, you only need to update the Neo4j credentials and the input .jsonl file path. After making these changes, the pipeline should run successfully.

Example Use Case: Highly Referenced Drugs, Locations, Organizations

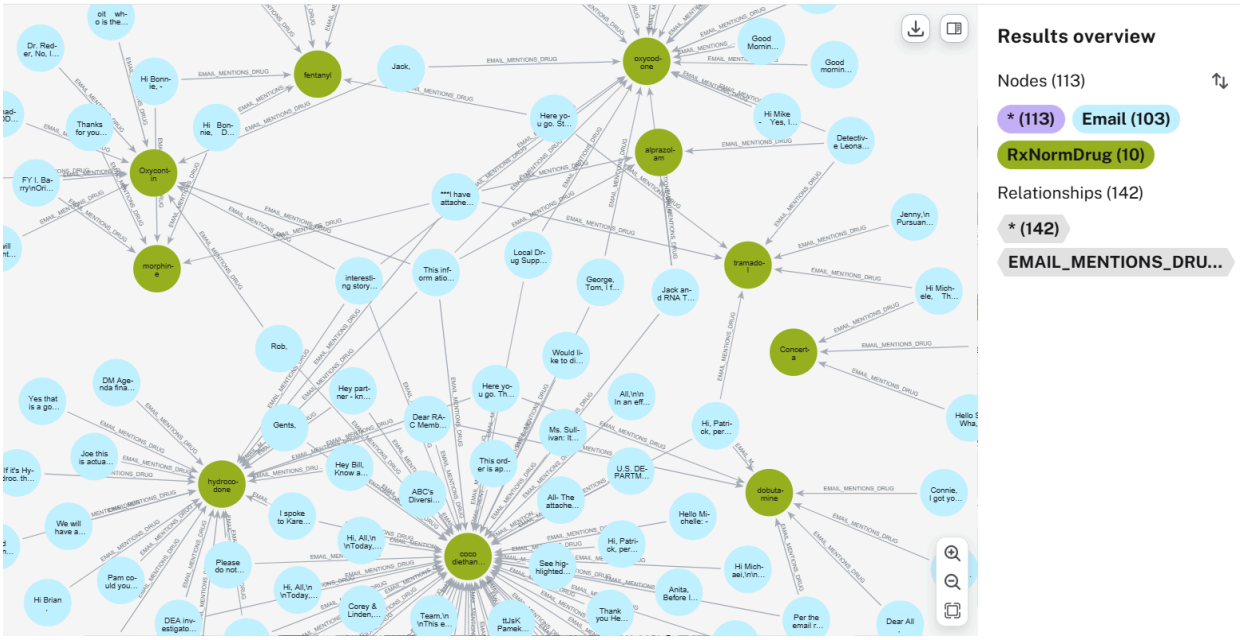

Figure 6: Top 10 Drugs Mentioned in Emails

We find the top 10 count of the drug names that are mentioned in the emails: coco diethanolamide, hydrocodone, oxycodone, Oxycontin, dobutamine, etc. Through their dense edge connectivity in the graph, these drugs are identified as important actors from internal email communications in discussions of concerns raised and of decision-making.

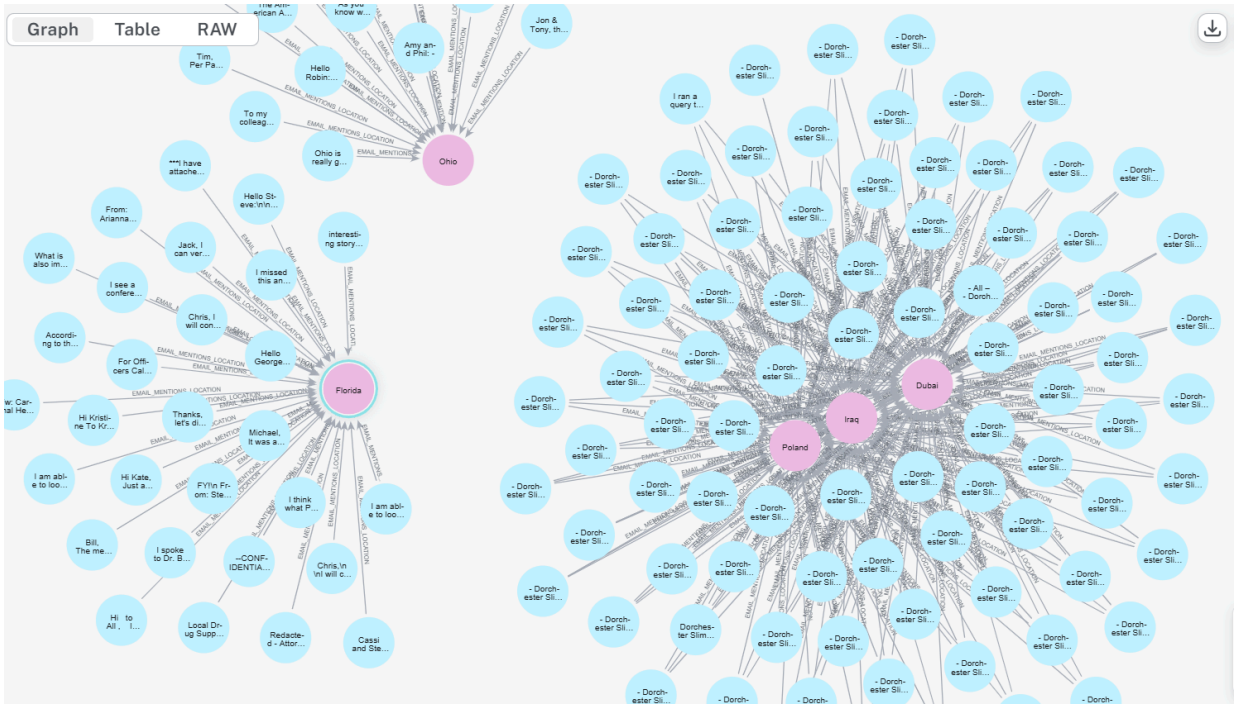

Figure 7: Top 5 Locations Mentioned in Emails

The next observation in continuation from the most frequently mentioned drug names is to find the most frequently mentioned locations as shown in Figure 7. Some locations frequently mentioned include Poland, Iraq, Dubai, Florida, and Ohio. This demonstrates direct linkage of specific sourcing, distribution, or operational grounds between drugs and locations. There is a path in the graph where the drug name nodes are connected to the location nodes through the email nodes. The locations themselves represent potential points of interest in the supply chain or corporate decision network. Emails that reference US states Florida and Ohio may signal operational or regulatory pressure points in domestic retail activity. This is particularly important in drug-related litigation, where several U.S. states serve as epicenters of opioid distribution. Additionally, mentions of places like Poland, Dubai, and Iraq point toward international supply or distribution routes.

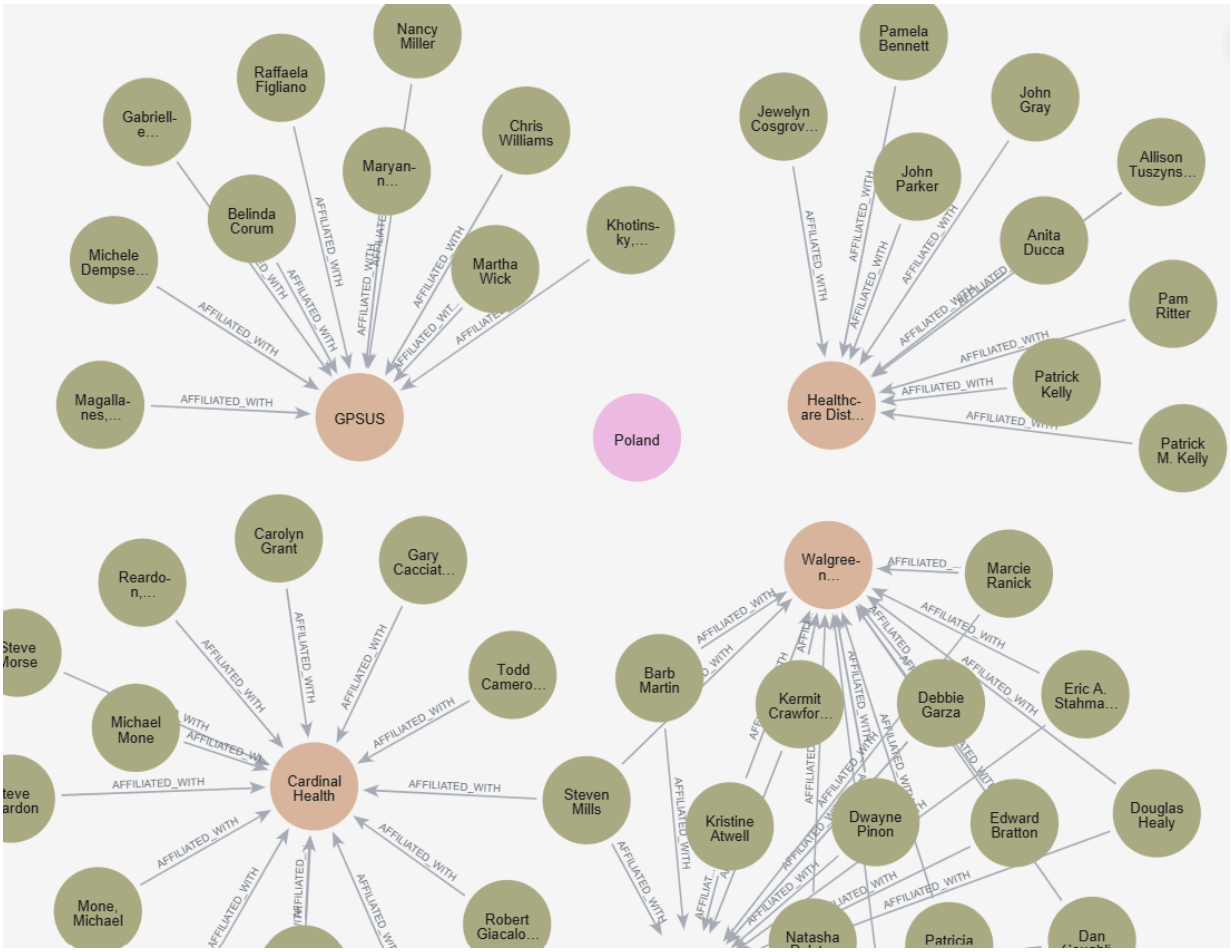

Figure 8: Poland's Affiliated Organizations and Their Personnel

The last piece of observation in this example use-case specifically focuses on Poland and its connected nodes. Figure 8 shows that organizations like GPSUS, Walgreens, and Cardinal Health have personnel who were actively engaged in email discussions involving Poland. This suggests that Poland is a relevant geography in its operational, regulatory, or commercial context and not an accidental mention. Cardinal Health, one of the largest drug distributors in the US, was named in multiple opioid-related lawsuits for failure to monitor suspicious orders, inadequate controls on controlled substances, and distribution practices into high-risk regions. Walmart, a major pharmaceutical chain, was accused in litigation of dispensing opioids without proper oversight, failing to implement compliance safeguards, and contributing to diversion and addiction pathways.

Future Development

For the future development of this project, we would like to continue enriching the semantic information and to scale the scope to more tables' email documents. We would also like to explore the attachments and render its content and add its information into our schema. We haven't looked at emails

in other languages, so the possible addition of some language data sources might be helpful. Even further down the line, there are more document types to explore and add onto our knowledge graph.

Looking at the opioid epidemic expressed in this project, it's possible to expand our dataset to include other public health issues like that of electronic cigarette companies that target flavored vapes towards teenagers. There could be a comparison drawn between the emails and other documents internally circulated within these drug companies that specifically make deceptive marketing claims across different decades. It's also possible to perform graph analytics and apply machine learning concepts to statistically evaluate the performance of a predictive model or decision-making model.