

Bridging Natural Language and Databases: A Knowledge Graph Approach to Text-to-SQL

Neo4j Meetup Tech Talk – May 2025

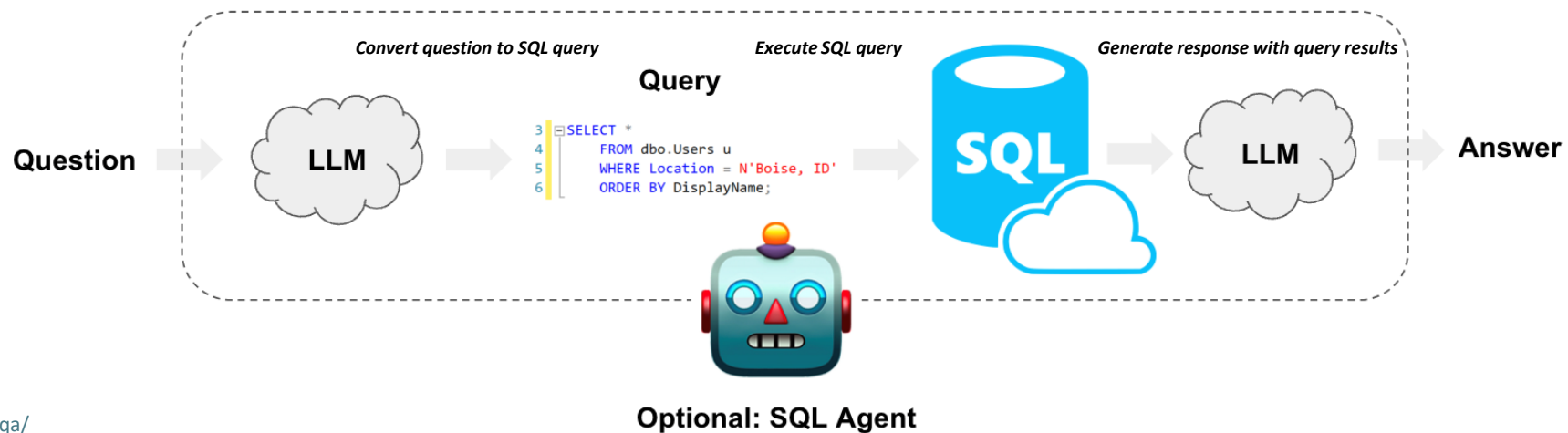
Kenneth Leung

Senior Data Scientist

Boston Consulting Group (BCG)

LLMs are enabling practical text-to-SQL for Q&A on enterprise data

- LLMs have enabled natural language access to structured data, making it possible to ask **questions** in plain language and receive accurate **SQL-backed answers**
- Text-to-SQL with LLMs allows **self-service analytics** without predefined dashboards or deep SQL expertise (e.g., knowledge of schemas and syntax)
- Huge impact size given **relational databases** remain the most popular data storage method



LLM-powered enterprise Q&A offers useful benefits across organizations

- **Removes dashboard dependency:** Business users can ask questions beyond prebuilt BI views
- **On-demand insights:** Reduces turnaround time by eliminating need to wait for new reports
- **Scales data access:** Makes complex data accessible to non-technical users in real time
- **Enhances decision-making:** Empowers staff with fast, flexible, accurate query capabilities
- **Bridges knowledge gaps:** "Always-available expert" familiar with schema and business domain

Model-generated SQL can pose security and data integrity risks; always scope database access narrowly to prevent unintended data exposure or harmful queries.

Using LLMs in isolation for text-to-SQL face several key limitations

- **Potential Hallucinations:** LLMs might fabricate information, providing responses that sound plausible but are factually incorrect
- **Opaqueness:** Difficult to determine clearly where and what sources the answer came from
- **Staleness:** Computational costs of LLM training makes it prohibitive to keep them updated
- **Incompleteness:** Due to probabilistic nature, LLMs may struggle to produce exhaustive answers

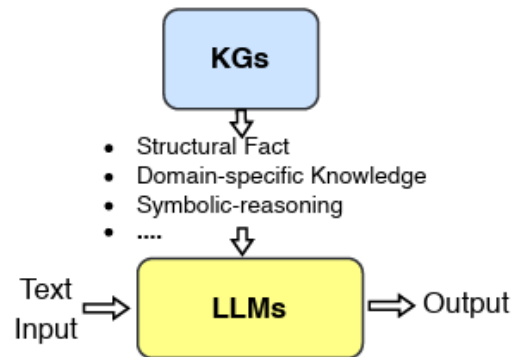
Even with schema access, LLMs may struggle with maintaining relevance amidst large prompt context, vague column names, limited semantic understanding, and missing domain-specific logic not captured in schema

Numerous barriers in enterprise data to overcome before applying text-to-SQL meaningfully

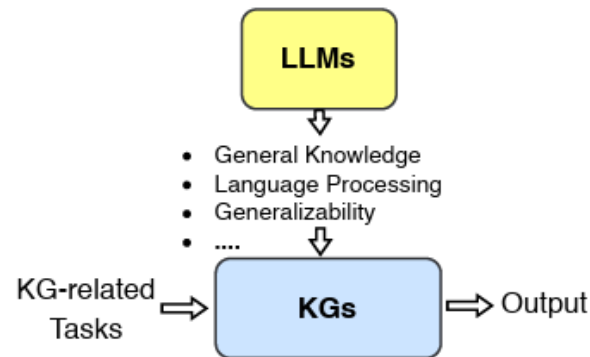
Barrier	Example
Fragmented and Inconsistent Schemas – Same domain concepts stored differently across diverse siloed databases with inconsistent table and column names	The same product may appear as different column names across tables e.g., <i>product_id</i> , <i>sku</i> , <i>product_code</i>
Hidden Relationships Between Data Values – Important mappings between values (e.g., codes, ID) often undocumented and only known through domain expertise	Score of 720 in <i>credit_score_table</i> means “Excellent,” but this meaning is not stored anywhere in the data
Limited Semantic Context in Data Dictionaries – Data dictionaries describe fields but do not always capture how data elements relate across tables	<i>transactions</i> table has column <i>channel</i> with values "W", "M", and "B", while <i>user_sessions</i> table uses "Web", "Mobile", and "Branch"
Unclear Join Paths and Table Connections - Related tables may lack clear join keys or metadata, making connections hard to identify	<i>loan_applications</i> table has <i>portfolio_id</i> , and <i>portfolio_details</i> holds related info — but no foreign key or metadata

A promising direction is to combine LLMs with knowledge graphs for text-to-SQL

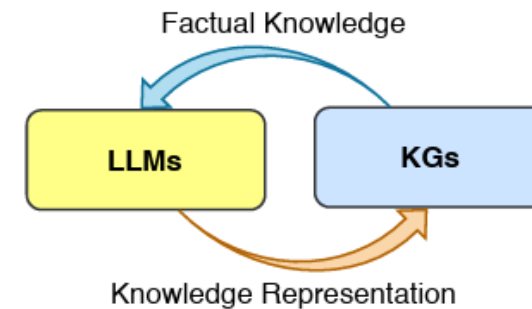
- Knowledge Graphs (KGs) act as an **external, structured reference**, offering complementary up-to-date domain-specific knowledge that LLMs were not trained on
- LLMs excel at language generation, while KGs provide trusted curated knowledge to fill business context gaps → Combining their flexibility and reliability **boosts transparency and accuracy**
- LLMs also helps extract or refine KGs from unstructured text, enabling a **virtuous cycle**



a. KG-enhanced LLMs

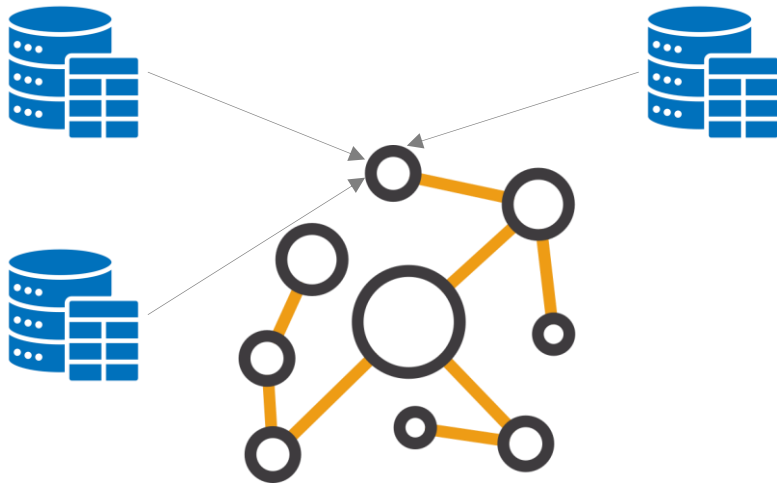


b. LLM-augmented KGs



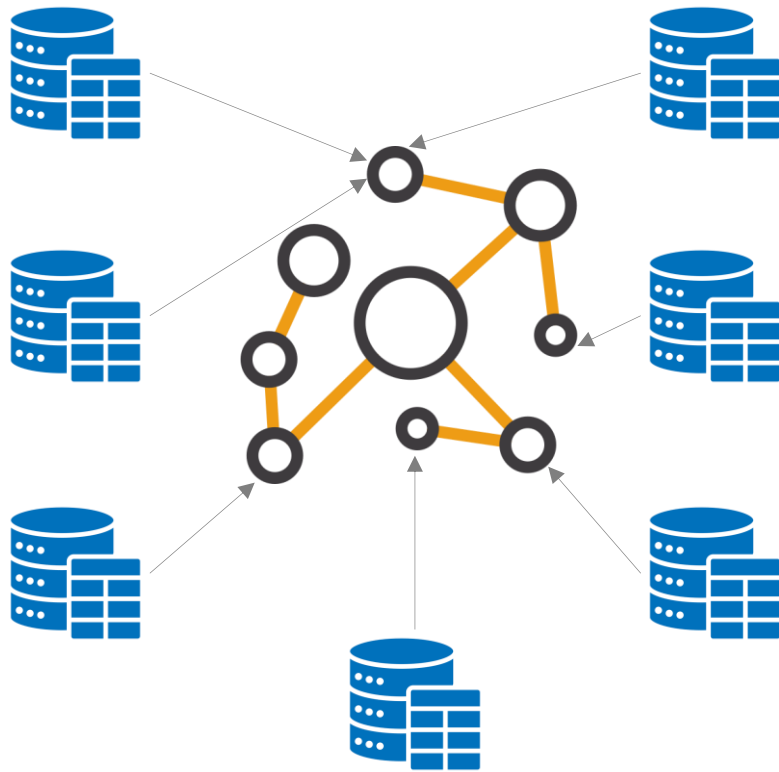
c. Synergized LLMs + KGs

Knowledge graph can serve as a semantic layer to unlock meaning in the data



- **Traditional relational databases** store data as rows and columns with relationships (e.g., customers, orders), but they **lack meaning** beyond structure
- A **semantic layer** sits on top of relational databases to add logic, linkages, and meaning by translating tables, rows and columns into **meaningful, connected concepts**
- For example, with a semantic layer:
 - customer_id mapped to **PERSON** concept
 - product_name & product_id mapped to **PRODUCT** concept
 - ordered_id connects **PERSON** to **PRODUCT** via a **PURCHASE**

A unified understanding of concepts in relational databases leads to many key benefits

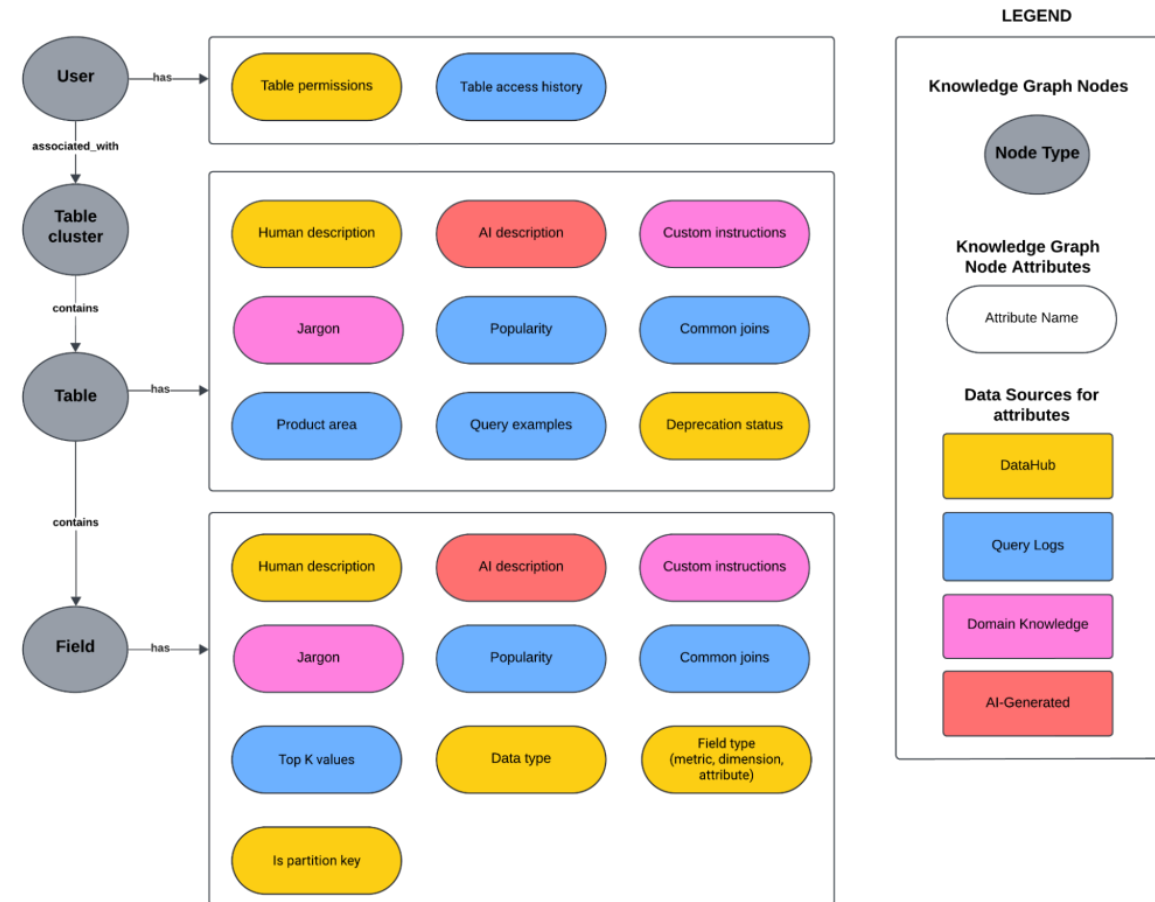


Building a well-defined understanding and **representation of enterprise data as knowledge graphs** leads to:

- Structured management of disparate data sources
- Better data quality, consistency, trust, and governance across enterprise systems
- Systematic capture of latent domain knowledge
- Consistent and reliable data and semantic interpretation across applications, reports, and teams
- Key foundation that improves text-to-SQL accuracy for querying and reasoning on relational databases

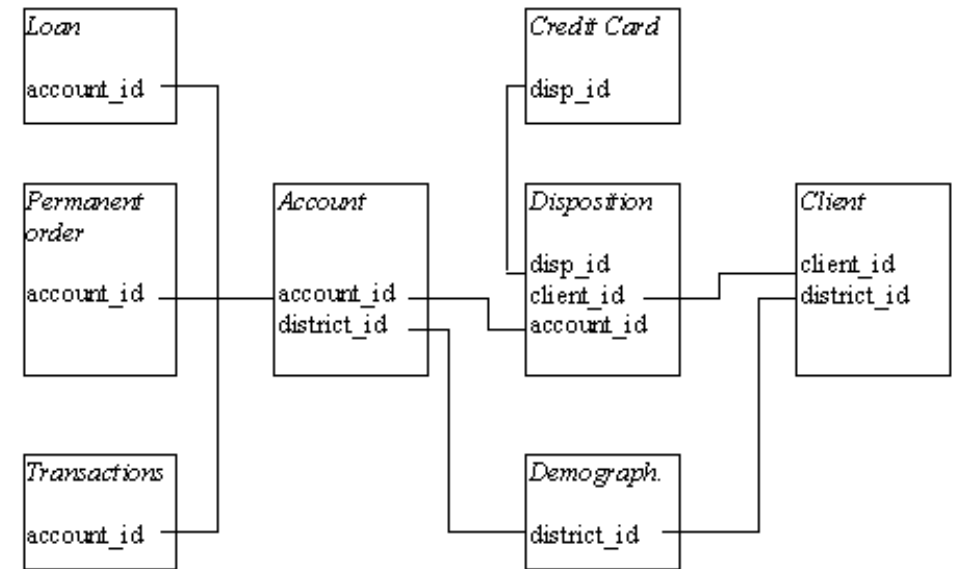
How LinkedIn uses knowledge graphs for practical text-to-SQL for in-house data analytics

- LinkedIn needed a deep semantic understanding of concepts and datasets for text-to-SQL
- They built a knowledge graph linking metadata, domain knowledge, and real query behavior
- LLMs use the KG to **rank the most relevant tables and fields during query generation**
- Combination of KG + LLM enables precise and context-aware SQL generation



Case Study – Context and Dataset

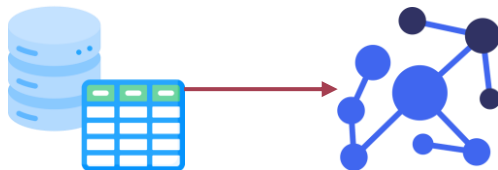
- Demonstrate build of a KG-based solution for text-to-SQL on financial data
- Using [Berka \(aka Czech financial\) dataset](#) (PKDD '99)
 - Contains account information, transaction histories, demographic data, loans, and client-product relationships from a Czech bank
 - Originally used to support tasks like customer segmentation, credit scoring, and churn prediction
 - Ideal due to its multi-table relational schema and real-world financial semantics and concepts



Methodology of Approach

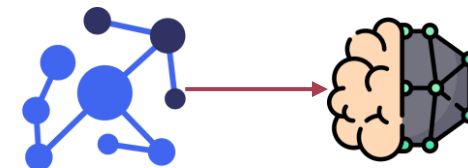
Part 1 – Indexing

- Convert SQL DB schema, data dictionary, and concept definitions into a structured semantic **knowledge graph** in Neo4j
- Result: **Semantic layer graph** built in Neo4j, representing logical meaning of SQL data



Part 2 – Inference

- Runtime process where user query is embedded and matched semantically against graph to support LLM-driven **SQL generation**
- Result: **Graph-enhanced SQL generation setup** using LLM + retrieval of structured knowledge



Part 1 of Approach – Indexing

Step 1 – Prepare knowledge sources to construct semantic representation of database

- Load configuration and metadata i.e., SQL database schema, data dictionaries, concept definitions (in YAML)



Step 2 – Initialize Neo4j driver and build graph mirroring relational structure and business meaning of database

- Ingest schema and concept data into Neo4j to form the nodes (Table, Column, Concept) and their corresponding relationships



Step 3 – Generate embeddings for Table nodes for similarity matching between text queries and tables

- Use embedding model to generate vector representations of **Tables** and their metadata e.g., title, remarks



Step 4 – Create vector and full-text search indexes in Neo4j

- Set up Neo4j indexes to enable fast, accurate retrieval
 - A vector index stores embedding vectors on nodes, allowing similarity search using cosine distance
 - A full-text index enables keyword-based search over descriptive fields.

Part 2 of Approach – Inference

Step 5 – Embed user's natural language query

- Convert query into an embedding using OpenAI's embedding model, to enable semantic similarity comparison with pre-embedded graph nodes (i.e., Tables)



Step 6 – Run hybrid retrieval on knowledge graph

- Utilizing vector search and full-text search, which is ideal for combining semantic understanding with literal keyword matching
- After identifying matching Table nodes, a Cypher query retrieves each table along with its columns, concepts those columns map to, and other columns in database that share those same concepts (akin to **GraphRAG**)



Step 7 – Augment prompt with retrieved knowledge

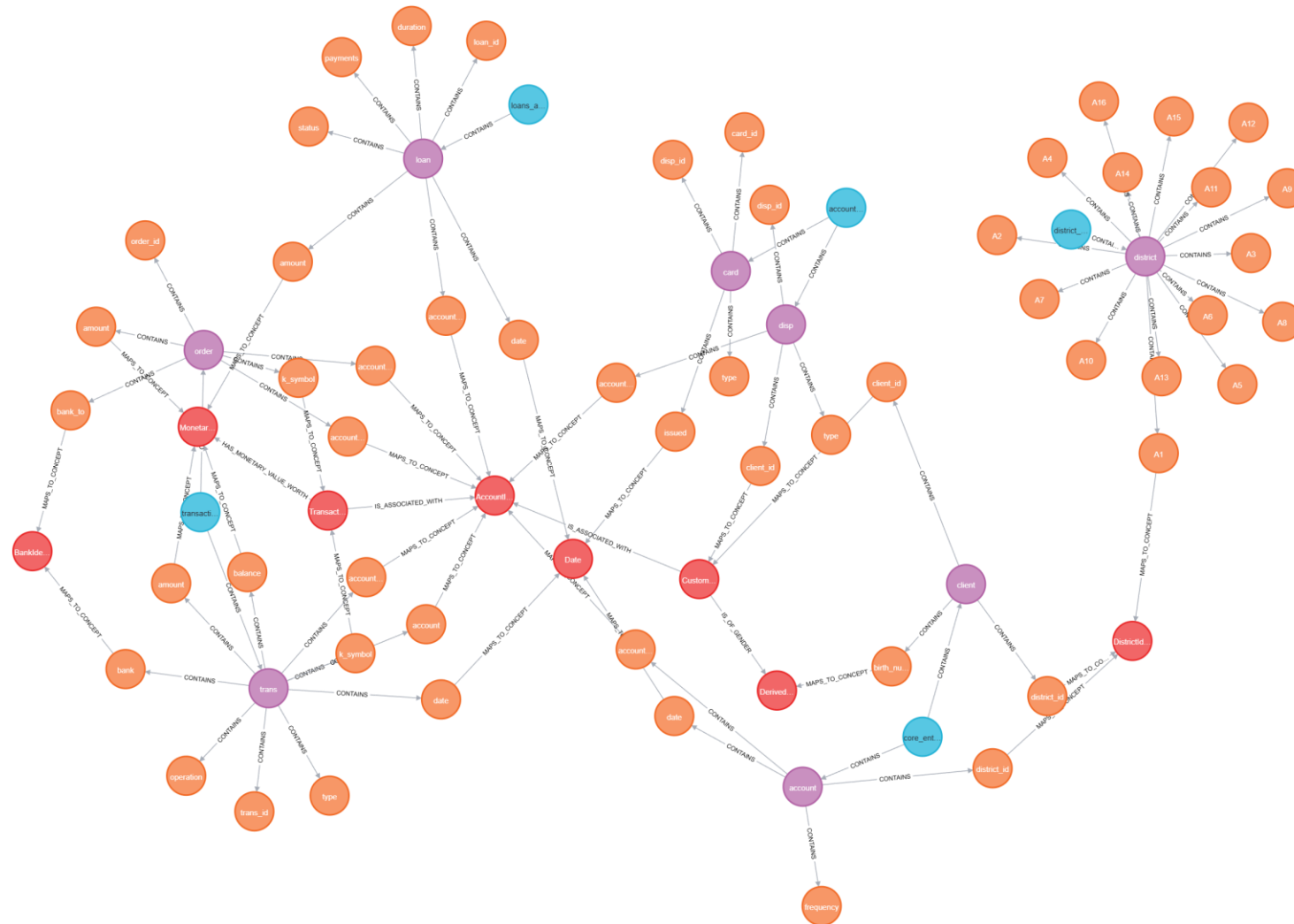
- Format and combine retrieved graph contents with original user query to build prompt for SQL generation



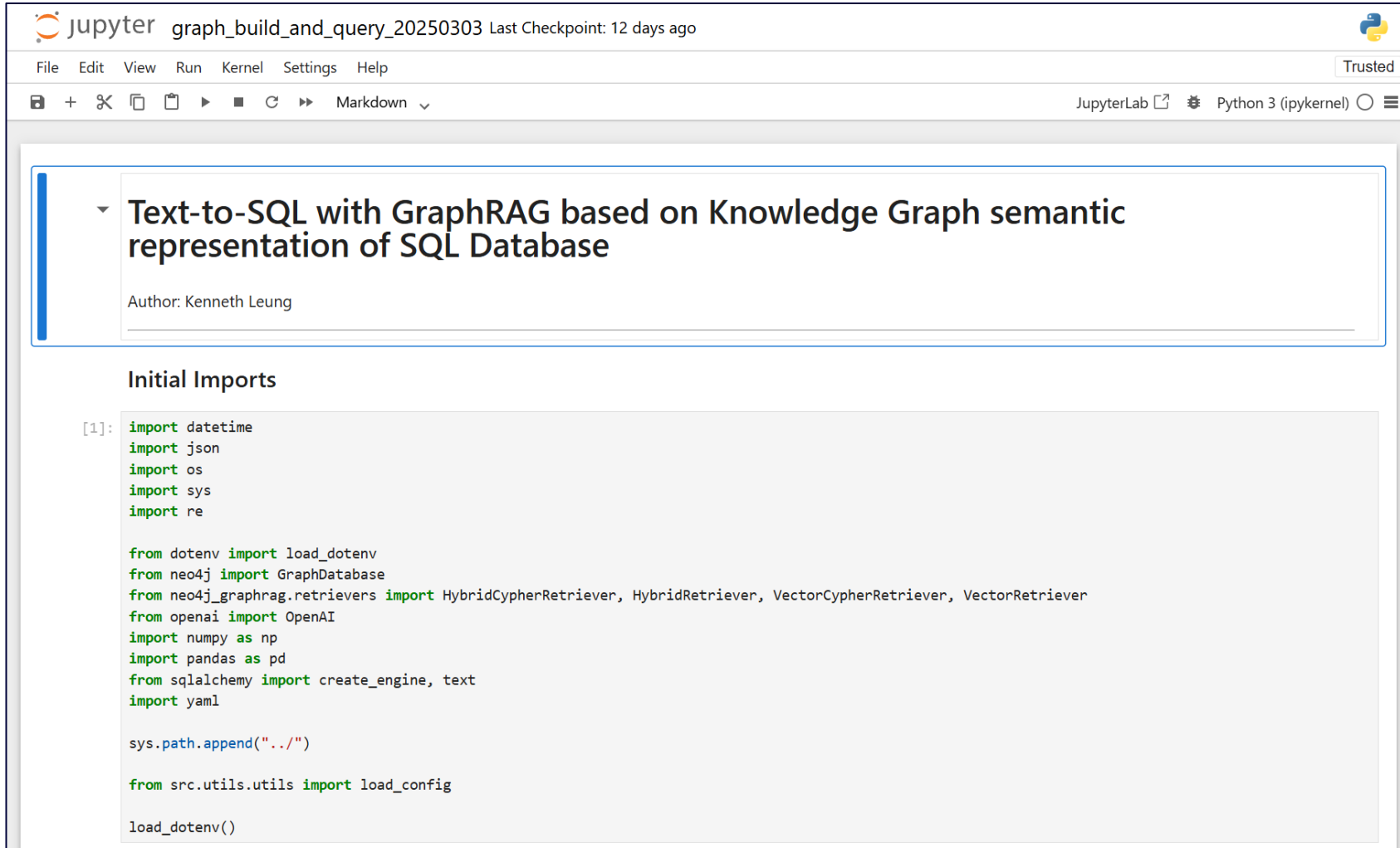
Step 8 – Generate SQL query via LLM with retrieved context

- Pass augmented prompt into LLM (GPT-4o-mini) to generate SQL query that answers user's question

Snapshot of Semantic Knowledge Graph in Neo4j



Code Walkthrough



The screenshot displays a JupyterLab environment. At the top, the header shows the Jupyter logo, the notebook name 'graph_build_and_query_20250303', and the last checkpoint time '12 days ago'. Below this is a menu bar with 'File', 'Edit', 'View', 'Run', 'Kernel', 'Settings', and 'Help'. A 'Trusted' status indicator is on the right. A toolbar with various icons for file operations and execution is located below the menu bar. The main area contains a notebook with a title 'Text-to-SQL with GraphRAG based on Knowledge Graph semantic representation of SQL Database' and an author 'Kenneth Leung'. The notebook content starts with a section 'Initial Imports' followed by a code cell [1]: containing the following Python code:

```
[1]: import datetime
import json
import os
import sys
import re

from dotenv import load_dotenv
from neo4j import GraphDatabase
from neo4j_graphrag.retrievers import HybridCypherRetriever, HybridRetriever, VectorCypherRetriever, VectorRetriever
from openai import OpenAI
import numpy as np
import pandas as pd
from sqlalchemy import create_engine, text
import yaml

sys.path.append("../")

from src.utils.utils import load_config

load_dotenv()
```

Takeaways

- **Building and combining a semantic knowledge graph** with hybrid retrieval gives the LLM structured, relevant context to make text-to-SQL more grounded and accurate
- By **separating schema and logic** into a knowledge graph, we reduce the burden on the LLM
- By representing fields with high-level **concepts**, the system can generalize and understand meaningful relationships across tables and schemas
- **There is no free lunch** – The process of knowledge engineering in setting up semantic layers still requires extensive manual effort in capturing and storing domain knowledge