Assignment 3: Applying Knowledge Graphs

Mater Student Evangelia P. Panourgia, f3352402
Professor Dr. Panos Alexopoulos

Structure Zip File

Before proceeding with our tasks, we will outline the structure of our zip file containing the following elements:

- folder: **json**: Contains all the output of our cypher queries used in our manually generated dataset. Each file inside it named with the format index_number.json.
- folder: **task1**: The file `binary_eval.txt` contains a manually assigned list of binary values (0s;represents "wrong" LLM-KG result and 1s represents "correct" LLM-KG result), which is used for the calculation of accuracy. In addition, it contains the file `report_llm.txt` the generated report produced by LLM-KG for our current prompt. The file *binary_eval_focused_on_cypher.txt* is related to an additional evaluation list based solely on errors related to cypher.
- folder: **task2**: Contains the files `zero_list.txt` contains a manually assigned list of binary values (0s;represents "wrong" LLM-KG result and 1s represents "correct" LLM-KG result), which is used for the calculation of accuracy ,`zero_shot_learning.txt` the generated report produced by LLM-KG for our current prompt for the zero shot learning. Furthermore, we have the file *zero_list_focused_on_cyphehr.txt* is related to an additional evaluation list based solely on errors related to the level of the generated cypher query and zero learning approach. Similarly, we have the related files for the few -shot learning being: `few_list.txt` `few_shot_learning_prompt_report.txt`, *few_shot_learning_focused_on_cypher.txt*
- **cypher_dataset.csv**: Contains our created manually dataset (more details in the section "Dataset File").
- *KG&LLMs@AUEB_Slides_Week4_GroundingLLMsWithKGs.ipynb***: Contains the whole code for our assignment.**

Dataset File

Before proceeding with our tasks, we will outline the structure of our generated dataset as referenced in the text. The dataset, named `cypher_dataset.csv`, contains the following columns:

- **main_category**: Indicates the primary category of our dataset, corresponding to the level of difficulty of the Cypher queries.
- **sub_category**: Represents the subcategory within our dataset, related to the types of queries we aim to test.
- **nlp_question**: The question formulated in natural language.
- **cypher_code**: The Cypher code designed to answer the nlp_question.
- **actual_returned_output**: The output returned from executing the Cypher code. It contains values of the format 2.*json* where its context is hosted on the folder json. This is because of the fact that some returns values had many rows and columns.

Task 1

To create our dataset of natural language questions suitable for the Neo4j movie graph, we first defined the **main categories** based on the **difficulty level** of each category. We then further divided each main category into **subcategories** according to the various types of queries we aimed to test. Regarding the main categories, in the box below, we observe the weighted values assigned as follows: 0.33 for "Simple Questions," 0.66 for "Advanced Questions," respectively, across each of the main categories.

Weighted equation for the "main" split of the generated dataset

$$GeneratedDataset = \mathbf{0.33}(SimpleQuestions) + \mathbf{0.66}(AdvancedQuestions)$$

We decided to distribute the weighted values this way because we expect the LLM to make more mistakes at the intermediate to complex levels of difficulty. To capture these errors effectively, we aim to generate more data for these categories, allowing us to extract "generalized" error patterns. In addition, we chose not to assign a high weight to the "Simple Questions," as there is a high likelihood that the LLM would exhibit an inflated performance. This is because simple questions do not accurately reflect the true complexity of Cypher queries. Instead, this weighting approach ensures the dataset remains "biased" towards intermediate to difficult cypher queries, reflecting a more balanced and realistic range of challenges. Both the main and subcategories of the analysis are illustrated in Figure 1. The **first level of the hierarchy** represents the **main categories**, while the **second level of the hierarchy** represents the **subcategories**. For each category, a percentage is shown in parentheses, indicating the proportion of data generated for that category. **The total number of generated dataset entries** is set at **27**. For instance, a percentage of (33%) for the first main category equates to 0.33×27, resulting in 9 rows generated for the main category *simple questions*. Similarly, for the subcategory *attribute queries*, a percentage of (11%) corresponds to 0.11×30, producing 3 rows for this subcategory. In the same manner, we interpret all percentages accordingly.

- **Simple Questions** (33.33%)
  - **Attribute Queries** (11.11%)
  - **Relationship Queries** (11.11%)
  - **Relationship Filter Queries** (11.11%)
- **Advanced Queries** (66.66%)
  - **Two Relationships Queries** (11.11%)
  - **Two Relationships with Attribute Filter Queries** (11.11%)
  - **Exclusions of Entities** (11.11%)
  - **Relationship with Aggregation Queries** (11.11%)
  - **Multi-Hop and Nested Relationship Queries (3 relationships)** (11.11%)
  - **Sub-Query &cardinality Queries** (11.11%)

Figure 1: Hierarchical Structure with Depth 2 for Cypher Query Categories

At this stage, we will analyze in depth the three main categories by examining the types of queries they encompass, which correspond to our subcategories. In other words, by explaining the components of each main category, we simultaneously provide an analysis of the associated subcategories.

Simple Questions either involve the retrieval of a single attribute from a node (e.g. What is the tagline of the movie The Matrix?, belonging to **Attribute Queries**) **or** involve a single relationship between nodes and return related nodes or attributes (e.g. Who wrote the movie A Few Good Men?, belonging to **Relationship Queries**) **or** involve the retrieval of a single attribute from a node using relation and filtering (e.g. What are the titles and release years of movies directed by Lana Wachowski that were released after 2003?, belonging to **Relationship Filter Queries**). In the file named *dataset_cypher.csv* the rows from 2 to 10 related to the category "Simple Questions".

Advanced Questions are a combination of intermediate and complex questions. They combine two different relationships to find multi-hop connections between entities (e.g. What is the name of the movie that Lilly Wachowski directed and in which Emil Eifrem acted?, belonging to **Two Relationships Queries**) **or** combine relationships with attribute-based filters which may be year or name (e.g. What movies did Rob Reiner direct and Aaron Sorkin write that were released after 1990?, belonging to **Two Relationships with Attribute Filter Queries**) **or** involve exclusion of specific entities from the results (e.g. What movies did Lana Wachowski direct that do not feature Keanu Reeves as an actor?, belonging to **Exclusions of Entities**) **or** apply basic aggregation (e.g. COUNT) within relationships to filter results (e.g. How many movies did each director direct between 2009 and 2020?, belonging

to **Relationship with Aggregation Queries**) **or** involve three relations (e.g. Which movies have different people as the reviewer, producer, and writer?, belonging to **Three Relationships Queries**) **or** involve **sub queries and** high-low **cardinality** (e.g. Which node has the lowest number of connections overall across all relationship types in the graph, and what is the relationship type associated with it?, this may be solved with multiple ways, wo in our prompt, we will provide example that solve it with the usage of sub query, belonging to the category **Sub-Nested Queries**).

Our general strategy for the whole creation of the dataset is described in the following bulletpoints:

- We engaged in an extensive dialogue with ChatGPT to generate potential queries for each main and subcategory. In some instances, additional guidance or follow-up questions were necessary, but it proved to be a valuable tool in enhancing the creativity and formulation of our sentences. In addition, it helped as with the generation of Cypher Queries.
- We validated each ChatGPT response by ensuring that the NLP question accurately corresponded to the appropriate main or subcategory and verifying the correctness of the Cypher query.
- With both the NLP question and the corresponding Cypher query generated by ChatGPT at our disposal, we proceeded to execute each query individually within the Neo4j workspace. This step was necessary to confirm the existence of data relevant to the current query.
- If no data was found, we further examined the provided graph (**via investigating the provided graph**) to identify the appropriate values for attributes, relationships, or nodes that contained data. We then adjusted our NLP question and Cypher query accordingly to ensure they were suitable. For example, if we wanted to adapt the value of the attribute related to the "title" of the movie, we run $MATCH(m : Movie)RETURNm.title$ in order to know all names of related titles or if we wanted to know beforehand all the possible combinations of people that acted in and and directed movies we firstly run the code in Figure 1.
- We repeated the process 27 times as 27 was the total number of rows of our generated datset.

Listing 1: Vice Versa process to retreive all movies in which two diffrent persons acted in and directed.

```
MATCH (director:Person)-[:DIRECTED]->(m:Movie)<-[:ACTED_IN]-(actor:Person)
WHERE director <> actor
RETURN m.title AS Movie, director.name AS Director, actor.name AS Actor
```

All of our subcategories contain exactly three rows, an intentional choice as this ensures an odd number of rows. This approach helps us avoid a 50% performance outcome per sub-category, allowing us to determine the trend or tendency of the LLM model's performance more clearly.

After creating our dataset, we implemented an architecture pattern that integrates a knowledge graph with a Large Language Model (LLM) to provide grounded answers. For the LLM, we utilized ChatOpenAI, and for the knowledge graph, we used Neo4jGraph. In the initial task, we passed the schema of the pre-defined Movies graph, along with each of our generated natural language questions, one at a time. The LLM processed the entire dataset, generating a Cypher query for each question, executing it, and then producing the final output as well as the internal process details. This approach allows us to understand the 'internal process' (that is the generated cypher query) behind each response. Due to the length of the generated output, the user is provided the option to open it in a text file. We leveraged this functionality by saving the complete context in a file named report_llm.txt, located in the 'task1' folder. This approach enables us to have access to both the 'internal process' followed by the LLM and the final output. Figure 2 illustrates the format in which each natural language question is stored. At this point, it should be mentioned that this functionality surely works in the case that the user run the Jupyter Notebook, locally, we don't know with sure if the same functionality is hosted on Google Colab.

- Index_Number , ==== NLP_Question===
- Generated Cypher: Cypher Query
- Full Context: The Output of the Generated Cypher Query
- Answer: where the "final LLM answer" or Error (e.g., syntax error)

Figure 2: High Level Structure of each NLP sentence in the Generated Report

With the generated report, we manually validated both the generated Cypher queries and their corresponding outputs against our created dataset. A part of our dataset contains the correct Cypher queries and outputs for

each NLP question, stored in JSON format in the $json$ folder. These were extracted from the Neo4j Aura console platform. To ensure consistency, we used a 'common key' for file checking: files in the $json$ folder follow the format ${index_number}.json$, while the file in the $task1$ folder is named $report_llm.txt$. The common key is the **Index_Number**, which ranges from 2 to 28 in increments of 1. We started with an initial value of 2 because our dataset, named $cypher_dataset.csv$ and located in the root folder, contains the first generated data in the second row, as the first row serves as the header of the CSV file. The aforementioned technique significantly enhanced the efficiency of our manual validation process. If both the Cypher query and the final output were correct, we assigned a value of 1 in a newly created column for our already created dataframe named df_cypher; otherwise, we assigned a value of 0. The name of the new binary column is "gpt_prediction" being helpful with the extraction and analysis of error patterns. So, our new dataset contains the already values being discussed at the beginning of this report (see section: Dataset File). Then, we filtered our dataset in order to analyze in depth the rows having value equal to 0 in column "gpt_prediction", being a total number of **14** rows errors.

We extracted the following types of errors:

- **Error in the level of Answer, returning "I don't know the answer"** (see Figure 2, but the Cypher query is correct). This phenomenon represents (**35.71%**) of the whole number of the detected errors. At this category belongs the indexes numbers: 13, 14, 15, 20, 22.
- **Error in the level of Answer, returning fewer data**, but the Cypher query is correct returning **all** needed number of data. This phenomenon represents (**14.28%**) of the whole number of the detected errors. At this category belongs the indexes numbers: 17, 18.
- **Error in the level of Generated Cypher** (see Figure: 2), where the Cypher query is wrong (e.g. taken incorrect attribute "born" instead of "released"). This phenomenon represents (**21.42%**) of the whole number of the detected errors. .At this category belongs the indexes numbers: 21, 23, 25.
- **Cypher Syntax Error**, this phenomenon represents (**28.57%**) of the whole number of the detected errors. At this category belongs the indexes numbers: 24, 26, 27, 28.

So, we can observe that about (1/2) of the errors belong to categories of errors related to the "Answer" of the query, and (1/2) with the error in the Cypher Queries. In the Figures : 2, 3, 4, **??**, **??**, we present the majority of errors related to the latter category of errors (that is number of indexes **21, 23, 25, 24, 26, 27, 28**). This in depth analysis is helpful, for defining examples in few shot learning in the next task.

Listing 2: Error in the level of Generated Cypher, Index Number 21

```
//How many movies did each director direct between 2009 and 2020?

//Correct Query (Mine)
MATCH (d:Person)-[:DIRECTED]->(m:Movie)
WHERE  m.released>2008 and m.released<=2020
WITH d, COUNT(m) AS highlyRatedMovies
RETURN d.name, highlyRatedMovies

//Wrong Query (ChatGPT Generation)
MATCH (d:Person)-[:DIRECTED]->(m:Movie)
WHERE d.born >= 2009 AND d.born <= 2020
RETURN d.name, COUNT(m) as numMoviesDirected
```

In Listing 2, we observe that LLM confused the attribute "born" with the attribute "released". In other words, the incorrect query attempts to filter the directors based on their birth year (`d.born`), which is irrelevant when determining the number of movies directed within the timeframe 2009 to 2020. Instead, the filter should have been applied to the release year of the movie nodes (`m.released`).

Listing 3: Error in the level of Generated Cypher, Index Number 23

```
//Which movies have different people as the writer, director, and actor?

//Correct query (Mine)
MATCH (w:Person)-[:WROTE]->(m:Movie)<-[:DIRECTED]-(d:Person)
MATCH (m)<-[:ACTED_IN]-(a:Person)
WHERE w <> d AND d <> a AND w <> a
RETURN DISTINCT m.title AS Movie, w.name AS Writer, d.name AS Director, a.name AS Actor

//Wrong Query (ChatGPT Generation)
```

```
MATCH (p:Person)-[:WROTE]->(m:Movie),
      (p2:Person)-[:DIRECTED]->(m),
      (p3:Person)-[:ACTED_IN]->(m)
RETURN DISTINCT m.title
```

In Figure 3, ChatGPT's query fails to ensure that the writer, director, and actor are distinct individuals, leading to potential cases where the same person is counted in multiple roles.

Listing 4: Error in the level of Generated Cypher, Index Number 25

```
//Which movies have different people as the actor, reviewer, and writer, and the actor was born
     after 1970?;

//Correct query (Mine)
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)
MATCH (m)<-[:REVIEWED]-(r:Person)
MATCH (m)<-[:WROTE]-(w:Person)
WHERE a <> r AND r <> w AND a <> w AND a.born > 1970
RETURN DISTINCT m.title AS Movie, a.name AS Actor, r.name AS Reviewer, w.name AS Writer

//Wrong Query (ChatGPT Generation)
MATCH (p:Person)-[:ACTED_IN]->(m:Movie)<-[:REVIEWED]-(p2:Person)-[:WROTE]->(m)
WHERE p.born > 1970
RETURN DISTINCT m.title
```

In Figure 4, Similarly, with the previous mistake, ChatGPT's query fails to ensure that the actor, reviewer, and writer are distinct individuals, potentially allowing the same person to fill multiple roles.

Now, we will present random 2 errors related to the syntax errors (that is index numbers 24, 26 because we want to belong to different sub categories that is Three Relationships Queries and Sub-Query & Cardinality Queries).

Listing 5: Error in the level of Generated Cypher, Index Number 24, subcategory: Three Relationships Queries

```
//Which movies have different people as the reviewer, producer, and writer?

//Correct query (Mine)
MATCH (r:Person)-[:REVIEWED]->(m:Movie)<-[:PRODUCED]-(p:Person)
MATCH (m)<-[:WROTE]-(w:Person)
WHERE r <> p AND p <> w AND r <> w
RETURN DISTINCT m.title AS Movie, r.name AS Reviewer, p.name AS Producer, w.name AS Writer;

//Wrong Query (ChatGPT Generation)
MATCH (m:Movie)
WHERE size((m)<-[:REVIEWED]-()) > 0
AND size((m)<-[:PRODUCED]-()) > 0
AND size((m)<-[:WROTE]-()) > 0
RETURN m.title

//syntax error
//Problem answering the question:  {code: Neo.ClientError.Statement.SyntaxError} {message: A
    pattern expression //should only be used in order to test the existence of a pattern. It can
     no longer be used inside the function //size(), an alternative is to replace size() with
    COUNT {}. (line 2, column 12 (offset: 27))
//"WHERE size((m)<-[:REVIEWED]-()) > 0 AND size((m)<-[:PRODUCED]-()) > 0 AND size((m)<-[:WROTE
    ]-()) > 0"
//             ^}
```

In Figure 5, ChatGPT's version fails due to a syntax issue with size() and does not guarantee distinct individuals for each role.

Listing 6: Syntax Error in the level of Generated Cypher, Index Number 26, subcategory: Sub-Query-Cardinality Queries

```
// Which node has the highest number of connections overall across all relationship types in
    the graph, and what is the relationship type associated with it?

//Correct query (Mine)
MATCH ()-[r]->() WITH DISTINCT type(r) AS relationshipType
CALL { WITH relationshipType MATCH (n)-[r]->()
WHERE type(r) = relationshipType
```

```
RETURN n AS Node, COUNT(r) AS connectionCount
ORDER BY connectionCount DESC LIMIT 1
}
RETURN relationshipType AS RelationshipType, Node.name AS NodeName, connectionCount AS
    NumberOfConnections ORDER BY NumberOfConnections desc limit 1

//Wrong Query (ChatGPT Generation)
MATCH (n)
WITH n, size((n)-[]-()) as connections
ORDER BY connections DESC
RETURN labels(n) as Node, connections, apoc.text.join([rel in relationships((n)-[]-()) | type(
    rel)], ", ") as Relationship
LIMIT 1;

//syntax error Problem answering the question:  {code: //Neo.ClientError.Statement.SyntaxError}
     {message: Type mismatch: expected Path but //was List<Path> (line 4, column 77 (offset:
    151))
//"RETURN labels(n) as Node, connections, apoc.text.join([rel in relationships((n)-//[]-()) |
    type(rel)], ", ") as Relationship"
                                                                                ^}
```

In Figure 6, ChatGPT's query incorrectly applies the relationships() function on a pattern instead of a valid path, resulting in a syntax error and failing to differentiate or count connections by relationship type.

Having a thorough understanding of the errors made by ChatGPT, we will present and evaluate performance of LLM for the given prompt. For this aim, we will calculate the **accuracy** of our LLM model that is the formula: **(Number of Correct Predictions) / (Total Number of Predictions)**. Note, in our case with the term prediction we mean the correct generated cypher and the correct final answer. We calculated that it is equal to about 0.48, meaning out of all the predictions, only 48% match the actual results, while the remaining 52% are incorrect. **This indicates that the model's performance is not much better than random guessing** (especially if it's a binary classification problem, where a random guess would have an accuracy of around 0.5). It suggests that the model may need further improvement or tuning to achieve better accuracy. Via applying more in depth analysis, we found that the model makes mistakes only in the main category named "Advanced Queries", which we waited intuitively. With an additional analysis we implemented, we found that the most dominant error **sub categories** are **Multi-Hop and Nested Relationship Queries(3 Relationships), Relationship with Aggregation Queries, Sub-Query-Cardinality Queries**. It is worth pointing out, that the first two sub categories are the "most" complex questions, and as a result it is quite logic that the model make mistakes. Bearing in mind all the evaluation analysis, and thanks to our clearly defined (sub) categories, we can say the following things to a possible user that want to run this LLM with our current simple prompt (see box: Simple Prompt):

- 1. The LLM (with our current promts) is "perfect" for "Simple Questions"
- 2. The LLM makes many mistakes in Queries related to 3 relationships (and more)
- 3. The LLM makes many mistakes in Queries related to high cardinality

Simple Prompt

CYPHER_GENERATION_TEMPLATE = """
Task:Generate Cypher statement to query a graph database.
Instructions:
Use only the provided relationship types and properties in the schema. Do not use any other relationship types or properties that are not provided. Schema: schema

Note: Do not include any explanations or apologies in your responses.
Do not respond to any questions that might ask anything else than for you to construct a Cypher statement.
Do not include any text except the generated Cypher statement.

The question is:
question"""

Note 1 : Each of [2] and [3] cover about 21% of the coverage of detected errors.

Note 2 : we did not mention the error of category related to teh sub category "Relationship with Aggregation Queries" because the majority of errors are due to the "final" answer and not due to incorrect generated cypher query.

Last but not least, via separating our dataset with main and sub categories, and applying in dept error analysis, it is quite helpful, for providing information for "dangerous" cases , "holes" of the model. So, if a potential user knows the "holes" then it will use our prompts only for the circumstances that is works well. In addition, the user knows the cases in which our model do not trained e.g. we did not passed NLP questions for recursion queries. Of course, our LLM need zero and few shot learning to be improved duw to the low calculated accuracy.

Task 2

Before delving into the process of prompt improvement, let's clarify the contents of Table 1, as this section is frequently referenced. First, recall that in Task 1, we recorded errors related to both Cypher query construction and final output accuracy. Specifically, errors in the Cypher queries included issues such as incorrect attribute usage, syntax errors, or improper filtering/conditions. In addition, we accounted for errors in the "final" output, such as cases where the Cypher query was correct, but the language model's response was incomplete (e.g., returning "I don't know the answer" despite the fact that the query returned data or providing fewer rows than expected).

In Table 1, we calculated accuracies for both types of errors separately. The first row, labeled "Overall Accuracy," reflects the accuracy considering all error types, while the second row, labeled "Cypher Query Accuracy," focuses solely on errors related to the Cypher query itself. The table's columns represent the various prompts used, including the basic prompt (a simple prompt passing only the task), the zero-shot prompt (related to zero-shot learning), and the few-shot prompt (related to few-shot learning).

	Basic Prompt	**Zero-Shot Prompt**	**Few-Shot Prompt**
Overall Accuracy	0.48	0.52	0.63
Cypher Queries Accuracy	0.74	0.74	0.81

Table 1: Accuracy for each case

Building on the analysis from Task 1, we aimed to improve the Cypher generation prompt. Our initial approach leveraged **zero-shot learning** (see "Zero Shot Learning Prompt") to provide precise guidance for generating accurate Cypher queries, explicitly avoiding examples. This approach focused on reducing syntax errors, incorrect properties, and incomplete outputs. In essence, we extended our basic prompts by providing additional instructions to our language model. From Table 1, we observe a slight improvement in overall accuracy, increasing from 0.48 to 0.52 compared to the basic prompt case. In contrast, the accuracy for Cypher queries remains stable across the different prompt variations (basic prompt and zero-shot prompt). Moreover, it is evident that eliminating errors associated with the final output significantly enhances the accuracy of our LLM. This indicates that most errors are **not** linked to Cypher generation. The related prompt is presented in the box below with title: Zero Shot Learning Prompt.

Additionally, we aimed to enhance our accuracy by implementing **few-shot learning**, providing "new" examples derived from our analysis in Task 1 to assess whether this approach would improve the model's performance. To achieve this, we created a prompt that includes examples for most of the recorded errors. We determined that an effective approach was to present the **question alongside the correct Cypher query** (instead of following a pattern question -> correct cypher query -> incorrect cypher query) and for the cases related to errors such as in the final output we provided guidelines and, in parentheses format, we also provide relevant examples. By examining the accuracy data in Table 1 and comparing the accuracy values between zero-shot and few-shot learning, we observed that our model demonstrates improved performance in both overall accuracy (increasing from 0.52 to 0.63) and the accuracy recorded to the case related to Cypher queries (rising from 0.74 to 0.81). Additionally, it is clear that minimizing errors associated with the final output substantially enhances the accuracy of our LLM, indicating that the majority of errors are **not** related to Cypher generation. **General notable comment**: Via filtering our data frame, we found that our LLM learned one of the difficult type of categories related to three relationships (sub category: Multi-Hop and Nested Relationship Queries (3 relationships)). The related prompt is presented in the boxes with titles: Few Shot Learning Prompt (part a') and Few Shot Learning prompt continuous (part b').

Due to time constraints and the labor-intensive process of manually labeling the performance of the LLM, we were limited in the number of prompts we could evaluate. The unique nature of our data, consisting of cypher

queries, made it challenging to automate the evaluation process reliably. However, given additional time, we would prioritize few-shot learning, as it demonstrated superior performance based on recorded accuracies, and we would provide additional examples to further enhance model evaluation. To clarify the future directions with greater specificity, we analyzed errors in our dataset focusing exclusively on Cypher errors within our optimal prompt, which uses few-shot learning. Our findings indicate that most errors fall under the category of Advanced Queries, specifically within the subcategories of **Exclusions of Entities** and **Sub-Query-Cardinality Queries**. This suggests a need to provide additional examples tailored to these types of queries to improve accuracy and reduce errors. Therefore, the future direction in **few-shot learning** with emphasis on questions "correct" cypher queries related to two aforementioned categories.

Zero Shot Learning Prompt

CYPHER_GENERATION_TEMPLATE = """ Task:Generate Cypher statement to query a graph database.
Instructions:
Use only the provided relationship types and properties in the schema.
Do not use any other relationship types or properties that are not provided.
Schema:
schema

Guidelines for Generating Accurate Cypher Queries and Responses:
- Provide Complete Output: Ensure your response includes the full output of the Cypher query, with no placeholder responses or partial answers.
- Retrieve All Relevant Data: Structure the Cypher query to return all required data points. Ensure the query is comprehensive and does not omit any needed information.
- Use Correct Properties and Filters: Reference properties and relationships exactly as defined in the schema, with careful attention to property names and required filtering conditions.
- Follow Cypher Syntax Strictly: Adhere closely to Cypher syntax to ensure that queries run without syntax errors.

Note: Do not include any explanations or apologies in your responses.
Do not respond to any questions that might ask anything else than for you to construct a Cypher statement.
Do not include any text except the generated Cypher statement.

The question is:
question"""

Few Shot Learning Prompt (part a')

CYPHER_GENERATION_TEMPLATE = """
Task: Generate a Cypher statement to query a graph database.

Instructions:
- Use only the relationship types and properties specified in the schema below.
- Avoid using any relationship types or properties not included in the schema.

Schema:
schema

Examples Cypher Queries

1. **Correct Use of the Attributes "born" and "released" Based on Node Type**

- **Question:** How many movies has each director directed after 2009?
- **Cypher Query:**
“‘ MATCH (d:Person)-[:DIRECTED]->(m:Movie)
WHERE m.released > 2009
RETURN d.name AS directorName, COUNT(m) AS directedMovies
“‘
- **Question:** Who are the individuals born between 1900 and 2020 that directed movies, and how many movies has each person directed?
- **Cypher Query:**
“‘ MATCH (d:Person)-[:DIRECTED]->(m:Movie)
WHERE d.born >= 1900 AND d.born <= 2020
RETURN d.name, COUNT(m) as numMoviesDirected
“‘
2. **Ensuring Different Individuals for Specific Roles for **three** relationships**
- **Question:** Which movies released after 2010 have a writer, director, and actor who were each born before 1980, and are all different individuals?
- **Cypher Query:**
“‘ MATCH (w:Person)-[:WROTE]->(m:Movie)<-[:DIRECTED]-(d:Person)
$MATCH(m) < -[: ACTED_IN] - (a : Person)$
WHERE w <> d AND d <> a AND w <> a
AND m.released > 2010
AND w.born < 1980
AND d.born < 1980
AND a.born < 1980
RETURN DISTINCT m.title AS Movie, m.released AS ReleaseYear,
w.name AS Writer, w.born AS WriterBorn,
d.name AS Director, d.born AS DirectorBborn,
a.name AS Actor, a.born AS ActorBorn
ORDER BY m.released DESC
“‘ """

Few Shot learning Prompt continuous (part b')

3. **Usage of subquery for Queries related to Cardinality**
- **Question:** For each distinct relationship type in the graph, which node has the lowest number of connections of that relationship type?
- **Cypher Query:**
“‘ MATCH ()-[r]->()
WITH DISTINCT type(r) AS relationshipType
CALL
WITH relationshipType
MATCH (n)-[r]->()
WHERE type(r) = relationshipType
RETURN n AS Node, COUNT(r) AS connectionCount
ORDER BY connectionCount ASC
LIMIT 1

RETURN relationshipType AS RelationshipType,
Node.name AS NodeName,
connectionCount AS NumberOfConnections
ORDER BY relationshipType;
“‘
Guidelines for Generating Accurate Cypher Queries:
- **Complete Output**: Ensure your response includes the full output of the Cypher query (e.g. if the cypher query returns 5 rows, don't answer "I don't know the answer").
- **Retrieve All Relevant Data**: Include all required data points in the query (e.g. if the cypher query returns 5 rows then in the final output display all of them).
- **Use Correct Properties and Filters**: Reference properties and relationships exactly as defined in the schema (e.g. understand the diffrence of attributes "born" vs "released").
- **Follow Cypher Syntax Strictly**: Adhere closely to Cypher syntax to avoid errors (e.g. don't use unsupported functions like ‘size()‘ with pattern expressions).

Important:
- Do not include explanations or apologies.
- Respond only to requests to construct a Cypher statement.
- Your response should include only the generated Cypher statement.

The question is:
question