SM3-Text-to-Query: Synthetic Multi-Model Medical Text-to-Query Benchmark

Sithursan Sivasubramaniam*, **Cedric Osei-Akoto***, **Yi Zhang**,
Kurt Stockinger, **Jonathan Fürst**†
Zurich University of Applied Sciences, Switzerland
{sivassit,oseiaced}@students.zhaw.ch, {zhay,stog,fues}@zhaw.ch

Abstract

Electronic health records (EHRs) are stored in various database systems with different database models on heterogeneous storage architectures, such as relational databases, document stores, or graph databases. These different database models have a big impact on query complexity and performance. While this has been a known fact in database research, its implications for the growing number of Text-to-Query systems have surprisingly not been investigated so far. In this paper, we present SM3-Text-to-Query, the first multi-model medical Text-to-Query benchmark based on synthetic patient data from Synthea, following the SNOMED-CT taxonomy—a widely used knowledge graph ontology covering medical terminology. SM3-Text-to-Query provides data representations for relational databases (PostgreSQL), document stores (MongoDB), and graph databases (Neo4j and GraphDB (RDF)), allowing the evaluation across four popular query languages, namely SQL, MQL, Cypher, and SPARQL. We systematically and manually develop 408 template questions, which we augment to construct a benchmark of 10K diverse natural language question/query pairs for these four query languages (40K pairs overall). On our dataset, we evaluate several common in-context-learning (ICL) approaches for a set of representative closed and open-source LLMs. Our evaluation sheds light on the trade-offs between database models and query languages for different ICL strategies and LLMs. Last, SM3-Text-to-Query is easily extendable to additional query languages or real, standard-based patient databases.

1 Introduction

The health sector is being increasingly digitalized, with data stored in electronic health records (EHR) [25]. In practice, those records can be kept in various forms and systems: (i) traditional relational databases such as PostgreSQL or Oracle implement the *relational data model* where data is organized into relations (tables) that are collections of tuples (rows). Users access data through the declarative query language SQL; (ii) popular document databases, such as MongoDB, model data as a collection of documents in the *document data model*, providing data access through specialized languages such as the MongoDB Query Language (MQL); (iii) graph databases (triple stores) model data as property graphs (e.g., Neo4j) or semantic RDF graphs (e.g., Ontotext GraphDB), providing interfaces through the query languages Cypher and SPARQL, respectively.

While the relational model and the SQL query language are still the primary choice for EHRs [22], there has been an increased interest in document and graph database models due to their schema flexibility and natural capacity to interconnect data sources across data silos [37, 11, 7]. E.g., AICCELERATE, a recent large-scale European pilot project on digital hospitals, uses a graph model

∗ Equal contribution.
† Corresponding author.

38th Conference on Neural Information Processing Systems (NeurIPS 2024) Track on Datasets and Benchmarks.

to connect various data sources, including traditional EHRs, other hospital data, wearables, and IoT devices [1]. Moreover, RDF graph databases that use SPARQL as the primary query language are widely used in life sciences, with medicine rapidly catching up [34].

The choice of database and the underlying core data model (relational, document, graph) has a large impact on read/write performance and query complexity. For example, the graph model naturally represents many-to-many relationships, such as connections between patients, doctors, treatments, and medical conditions, whereas relational databases require potentially expensive join operations and complex queries. Document databases have only rudimentary support for many-to-many relationships and aim at scenarios where data is not highly interconnected and stored in collections of documents with a flexible schema [19]. Figure 1 shows an example of a single user question together with the corresponding query statements in four query languages. While these differences have been a known fact in database research and industry, *its implications for the growing number of Text-to-Query systems have surprisingly not been investigated so far.*

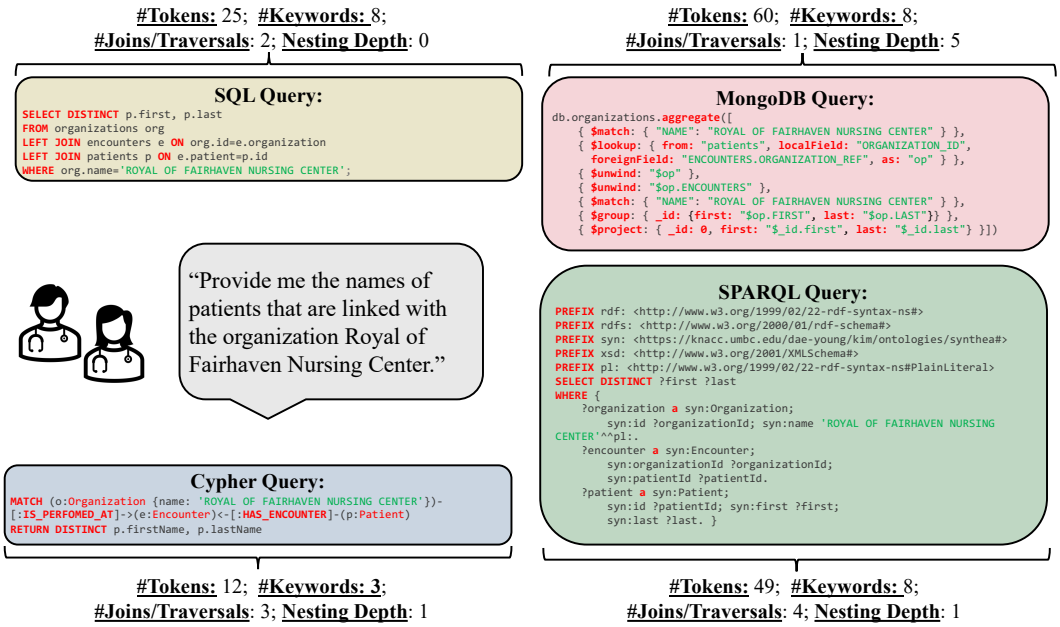

Figure 1: Differences across query languages and database systems for the same user request.

Text-to-Query systems have seen a recent growth in the number of developed methods and new high scores, mainly due to the transformer architecture and advances in Large Language Models (LLMs) [17]. The idea is compelling: Instead of writing database queries, users express their intends in natural language and a Text-to-Query system with access to the underlying database translates them into valid query statements (e.g., SQL or SPARQL). This is especially relevant in a digital health context in which the domain experts (e.g., nurses, doctors, or admin staff) cannot be expected to write complex queries and currently rely on pre-defined rule conversion systems that limit the potential questions that can be asked [22].

Existing Text-to-Query datasets and benchmarks have usually been focusing on single database models and query languages. E.g., Spider [42], WikiSQL [44], BIRD [23], ScienceBenchmark [43], Statbot.swiss [29] and EHRSQL [22] (Text-to-SQL) and LC-Quad 2 [9] and Spider4SPARQL [20], for Text-to-SPARQL, also called Knowledge Graph Question Answering (KGQA). Only recently, in FootballDB [12], have there been initial attempts to evaluate the data model impact inside a single database system (with different schemas for the same data). Further, there has been initial works across two database models and query languages. E.g., a recent comparison of SQL and SPARQL based on the MIMICSQL dataset [40] finds a 34-point accuracy difference [31].

However, *no existing work in and outside the medical domain enables the evaluation of Text-to-Query across multiple core database models and query languages*. This is the goal of our dataset and benchmark. In this paper, we present *SM3-Text-to-Query, a first Multi-Model Medical Text-to-Query Benchmark based on synthetic patient data*. SM3-Text-to-Query provides the following key features:

- **Standard-based and privacy-preserving.** SM3-Text-to-Query has been carefully constructed from Synthea [39], a synthetic patient data simulator. Thus, there are no privacy implications in the published dataset. In our data schemas, we follow the SNOMED-CT taxonomy [35], a commonly used medical knowledge graph ontology that is widely applied across institutions and countries, enabling interoperability of health data. *This ensures the wide impact of the benchmark on real-world health use cases.*
- **Three database models—four database systems and query languages.** Involving database experts in the process, we design four data representations (schemas) in PostgreSQL, MongoDB, Neo4j, and RDF (GraphDB), and create transformations for them from the Synthea output. These databases represent three core database models: relational, document, and graph model. The databases with *the same data content* can be accessed through four different query languages: SQL, MQL, Cypher, and SPARQL. SM3-Text-to-Query is, to our knowledge, the first benchmark with these features. *Our chosen database systems and query languages constitute a wide representation of the most popular systems according to DB-Engines Ranking [36].*
- **Systematic and expandable question generation.** We systematically and manually create a set of 408 template questions covering the major entities and properties of the Synthea data. These template questions are then automatically enhanced and enriched to result in a benchmark set of 10K natural language/query pairs for each query language. The enrichment is performed via parameterizable sampling methods to retrieve, for instance, patient names or allergies from the underlying database. Our method is easily extendable through the addition of new templates (e.g., different languages, different questions, paraphrasing) or through plugging in real patient databases modeled according to SNOMED-CT. *This ensures the benchmark is future-proof and can be adjusted to other use scenarios.*

2 Related Work

In this section, we review related works for Text-to-Query systems with a focus on (i) medical data and (ii) generally relevant benchmarks.

Medical focus. MIMICSQL [40] is derived from the MIMIC-III database [15], containing 10K unique questions tailored to medical quality assurance tasks. To avoid potential limitations, such as fixed question structures, MIMICSQL underwent a filtering and paraphrasing process performed by expert freelancers. MIMIC-SPARQL [31] builds on the framework of MIMICSQL [40] and customizes its question templates to query a modified schema with SPARQL. With a similar structure to MIMICSQL, it provides 10K unique questions tailored to medical QA tasks. Last, EHRSQL [22] provides a benchmark for text-to-SQL tasks with a focus on electronic health records (EHR). It is based on MIMIC-III and eICU databases, while the 230 question templates are derived from user surveys. Based on these 230 templates, EHRSQL generates 24K questions/query pairs for SQL.

General benchmarks. The WikiSQL [44] dataset is a well-known general Text-to-SQL benchmark that comprises over 80K text/SQL pairs. What makes this dataset noteworthy is the wide distribution of queries over 24,241 tables. Spider [42] is considered one of the most popular cross-domain text-to-SQL datasets and consists of 10,181 questions with 5,693 unique SQL queries on 200 databases. KaggleDBQA [21] builds on large-scale datasets such as Spider and WikiSQL to provide a cross-domain dataset with domain-specific data types. BIRD [23] is a comprehensive resource for question answering (QA) that includes 12,751 unique questions from various repositories such as Kaggle, CTU Prague, and Open Tables and covers 37 subject areas. BIRD targets real-world applications by including complex examples from 95 large databases totaling 33.4 GB. ScienceBenchmark [43] presents three real-world, domain-specific text-to-SQL datasets. In comparison to other datasets, it reflects the high importance of domain-specific benchmark datasets for real-world text-to-SQL tasks. Last, FootballDB [12] investigates different database schemas and their impact on Text-to-SQL systems inside a single database. The questions are derived from a real deployment with end users.

Compared to these works, the main novelty of SM3-Text-to-Query is that it is, to the best of our knowledge, the first dataset and benchmark that allows for the evaluation of Text-to-Query systems across three core database models (relational, graph, document) and four query languages (SQL, SPARQL, Cypher, MQL). FootballDB [12] is comparable in terms of analyzing the schema dependency of Text-to-SQL systems inside a single database. However, it only targets SQL, a single

database model (relational), and query language (SQL). For our template-based approach, we take inspiration from EHRSQL [22]. We complement their idea with synthetic data generation following a widely used medical standard (SNOMED [35]). This makes our benchmark relevant to digital health scenarios worldwide, where databases follow the SNOMED medical naming taxonomy.

3 SM3-Text-to-Query Benchmark Construction

Our SM3-Text-to-Query benchmark construction consists of two main steps. First, we construct the database based on synthetic medical data in four data models (Figure 2). Second, we implement a template-based text/query-pair construction approach. The dataset was created over a period of more than a year in the context of a project with health professionals from two university hospitals, medical doctors, nurses, data scientists, and computer scientists who have been working in the respective fields for 5+ years. We constructed the question templates to cover all SNOMED CT entities. The database queries were mainly written by two undergraduates and one PhD student with a computer science and database background and verified by two faculty members.

3.1 Database Construction

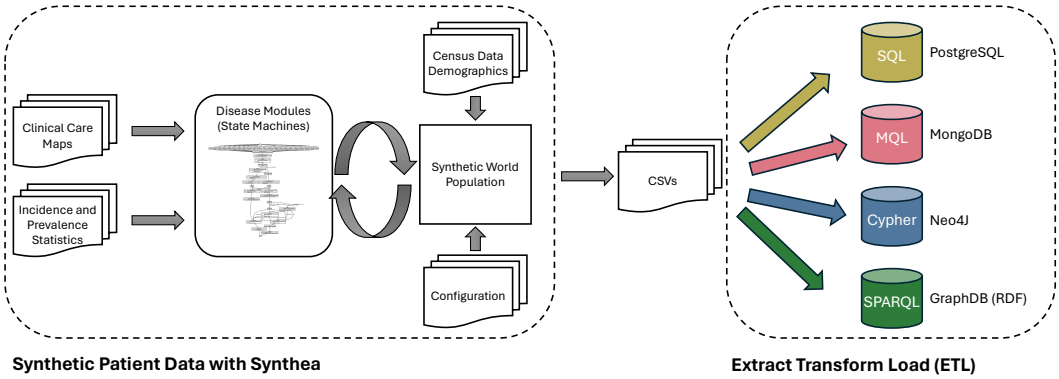

Figure 2: **Database construction from Synthetic Patient Data**. Synthea uses clinical care maps and statistics to build models of disease progression and treatment, encoded as state transition machines. The synthetic world population is seeded with census data demographics and configuration options, enabling the creation of realistic patient data in 18 classes, which we export as CSVs. We implement custom Extract Transform and Load (ETL) pipelines to transform these CSVs to four database systems and models.

3.1.1 Synthetic Patient Data with Synthea

As much of this data only reaches its full potential when it is interoperable across organizations, such as hospitals, insurance providers, and specialists, there has been a move toward standardization in healthcare. Here, SNOMED Clinical Terms (CT) is considered to be the most comprehensive, multilingual clinical healthcare terminology in the world [5]. While there exists a range of clinical EHR datasets such as MIMIC-III [15] and eICU [32], they are based on de-identified data from single countries (eICU) or even just single medical centers (MIMIC-III) and do not follow the SNOMED CT taxonomy.

To construct the SM3-Text-to-Query benchmark, we, therefore, choose to build on synthetic but standardized data that we generate through Synthea [39]. Synthea is an open-source, synthetic patient generator that models the medical history of synthetic patients and their electronic health records while being compatible with SNOMED CT. Its generated data includes 18 classes representing various aspects of healthcare, such as allergies, care plans, and medications, and is available in CSV format, FHIR, C-CDA, and CPCDS formats [26]. Synthea employs top-down and bottom-up approaches to generate structured synthetic EHRs throughout a patient's life. E.g., the simulation incorporates models for the top ten reasons for primary care visits and chronic conditions responsible for the most years of life lost in the United States, based on US Census Bureau, Centers for Disease

Control and Prevention, and National Institutes of Health reports. Further, international populations can be simulated through provided metadata and configuration files for currently 25 countries [27].

3.1.2 Data Transformation to different Databases

Based on the Synthea output, we define data schemas/ontologies and implement Extract, Transform and Load (ETL) pipelines for our chosen databases: PostgreSQL (SQL), MongoDB (MQL), Neo4j (Cypher), GraphDB (RDF). For PostgreSQL, we define appropriate data types and consistency constraints (mainly primary and foreign keys). MongoDB, as a document database, does not enforce a strict schema as relational databases. Here, we define a JSON schema following a tree structure with four top-level collections (patients, organizations, providers, payers) and the remaining entities being embedded in these collections. We connect documents across collections through ID references (`$lookup` operator as equivalent to Join). For Neo4J, we implement ETL by following the guidelines provided by Neo4j Solutions [14], adapting it due to missing classes and connections in the original configuration files. For RDF, we extend Synthea-RDF [26] to cover all Synthea attributes and automate the conversion of data from CSV files into RDF as Terse RDF Triple Language (TTL). All four data models can be found in Appendix A.6 and in the supplement material.

3.2 Text/Query-Pairs Construction

To construct a dataset of text/query-pairs, we follow the established template-based approach [10, 40, 31, 22] in which a set of template questions is augmented with values to scale the dataset without extensive manual efforts. Together with the standard-based Synthea data (SNOMED CT taxonomy), our generation process has the following advantages:

- **Coverage and diversity through Synthea data generation.** Through the use of templates, the benchmark dataset can be automatically adjusted to different Synthea datasets (e.g., for different patient populations). Each template is tagged with the relevant Synthea entities (e.g., patient, claim). Based on this structure, our method allows for the construction of datasets that only cover a subset of entities (e.g., only focusing on an insurance database).
- **Reduced bias in machine learning methods of the task.** By filling the query templates with parameterizable values, various biases of text-to-query methods can be exploited. Thus, we can avoid the respective LLMs overfitting.
- **Standardized evaluation over different systems.** The creation of standardized templates is possible through the implementation of the SNOMED terminology in the Synthea dataset. The benchmark dataset leverages SNOMED attributes, i.e., standardized medical terminology, for the evaluation of queries over different systems and database models. The same template questions can easily be combined with real-patient data following SNOMED medical terminology.

Overall, we create 408 template questions (see supplement material for a full list and Appendix A.4 for an example) in a structured way that is guided by the goal to cover all 18 entity types created by Synthea. The template questions include WH* and non-WH questions, factual questions, linking questions, summarization questions, and cause-effect questions. We tag each template question with its related entities and question type.

Last, following previous work [22], we define 10 non-answerable medical and 5 non-medical questions. These are questions that cannot be answered from data stored in the respective databases but would require additional information. For instance, *What is the marital status of patient Max Müller?*

For each question template, we manually develop the corresponding query in SQL, SPARQL, Cypher and MQL. The queries are then verified by a second expert for correctness. For scaling the template questions, we augment them by automatically inserting values such as IDs, descriptions of diseases, and patient names queried from the database. This data augmentation step is fully configurable and can be used to generate enriched and linguistically diverse text/query pairs for arbitrary Synthea databases and in-the-wild databases following the SNOMED CT standard.

*Who, What, Where, When, and Why.

4 Dataset Analysis and Comparison

Question and Query Statistics. For SM3-Text-to-Query, we use our method described in Section 3.1 to construct a synthetic multi-modal dataset of 10K text/query pairs for each of the four query languages (resulting in 40K individual samples). The dataset is based on the default Synthea configuration with medical records of 100 living and 10 diseased patients. We split the data into 6K train, 2K dev, and 2K test with stratified sampling where the strata are the entity types that the questions are tagged with. Figure 3 summarizes dev and test, with the distribution of different types of user questions on the left. While our template questions cover all SNOMED entities, there exist more templates for allergies, imaging studies, patients, and payers, which is why they are over-represented in the augmented dataset.

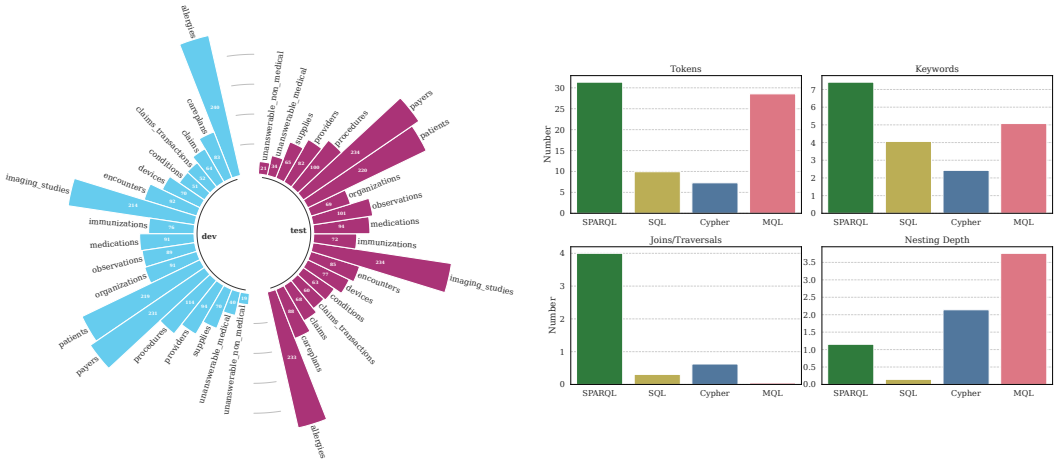

Figure 3: Dataset distribution for dev and train (left); Query statistics for query languages (right).

While all queries across query languages return the same information for a single user question, they vary greatly in their characteristics (Figure 3, right). For instance, SPARQL queries have the largest number of tokens and keywords, followed by MongoDB's MQL queries and SQL queries, while Cypher queries are the most compact ones. We also count the number of joins/traversals and the nesting depth (based on opening/closing characters) of the queries, an established complexity measure in programming language research [3]. Here, MQL requires notably the fewest joins (`$lookup` operators) while exhibiting the highest nesting depth (due to the embedded collection structure in its schema).

Comparison to other datasets. Table 1 summarizes SM3-Text-to-Query with respect to a selection of relevant datasets and benchmarks. SM3 has a similar number of example questions compared to other datasets, while the number of corresponding queries is substantially higher due to the translation to four query languages. Qualitatively, *SM3 is the only dataset that enables an evaluation across four different query languages, which is not only unique for medical data but does currently not exist for any domain.* Last, SM3 is standard-based (SNOMED), making it compatible with standard-based real health databases and extensible through our template-based approach.

Table 1: A comparison between SM3 and other relevant Text-to-Query datasets and benchmarks (NA-Qs: non-answerable questions; Standard: standard-based, e.g., SNOMED; Template: Template-based, which allows the dataset to be easily extended).

Dataset	Data source	# Questions	# Queries	NA-Qs	Medical	Standard	Template	SQL	MQL	SPARQL	Cypher
MIMICSQL [40]	MIMIC-III [15]	10,000	10,000	✗	✓	✗	✓	✓	✗	✗	✗
MIMIC-SPARQL [31]	MIMIC-III [15]	10,000	10,000	✗	✓	✗	✓	✗	✗	✓	✗
EHRSQL [22]	MIMIC-III [15], eICU [32]	24,000	24,000	✓	✓	✗	✓	✓	✗	✗	✗
BIRD [23]	Kaggle, CTU Prague, Web	12,751	12,751	✗	✗	✗	✗	✓	✗	✗	✗
SM3 TEXT-TO-QUERY	Synthea [27]	10,000	40,000	✓	✓	✓	✓	✓	✓	✓	✓

Comparison of query complexity. We also compare SM3 query complexity against the two common medical Text-to-SQL datasets EHRSQL [22] and MIMICSQL [40] (Table 2). EHRSQL and MIMICSQL feature more complex SQL queries, such as temporal ones. However, the strength of our dataset lies in its cross-model aspect, which includes Cypher, MQL, and SPARQL. Our MQL

and SPARQL queries have similar complexity in terms of token count. SPARQL even includes more joins and traversals than existing benchmarks, while MQL has more nesting.

Table 2: SM3-Text-to-Query query statistic compared to recent medical Text-to-SQL datasets.

	EHRSQL	MIMICSQL	$\text{SM3}_{\text{Cypher}}$	SM3_{MQL}	$\text{SM3}_{\text{SPARQL}}$	SM3_{SQL}
Tokens	**32.63**	19.45	7.28	28.56	31.35	9.93
Keywords	**11.74**	6.38	2.43	5.08	7.41	4.06
Joins/Traversals	0.33	0.64	0.62	0.04	**4.00**	0.31
Nesting Depth	1.37	0.77	2.14	**3.77**	1.15	0.15

5 Baseline Experimental Evaluation

The goal of our experiments is to evaluate how well large language models (LLMs) perform in translating natural language questions into four different query languages provided by our novel benchmark. We restrict ourselves to LLMs as these models dominate the leader boards of popular Text-to-Query benchmarks such as Spider [42].

5.1 Experimental Setup

For our baseline experiments, we select a set of four common open and closed-sourced LLMs and implement the same in-context learning (ICL) prompting strategies across our four query languages. Here, we follow standard practices in terms of task instruction formulation and the inclusion of schema/ontology information as well as few-shot examples. We take inspiration from Nan et al. who propose general design strategies for enhancing Text-to-SQL capabilities in LLMs [28], widely adopted in recent applications. Further, Chang and Fosler-Lussier demonstrate the importance of including database schema and content [6]. Liu and Tan suggest using notes and annotations to mitigate unexpected behaviors of LLMs [24], improving accuracy. Drawing upon these strategies, we developed various prompt templates tailored to the requirements of our experimental settings. The detailed templates applied in the experiments are listed in Appendix A.5.

As open-source LLMs, we select Meta Llama3-8b and Llama3-70b (instruction-tuned variants) [2]. As closed-source LLMs, we select Google Gemini 1.0 Pro [38] and OpenAI GPT-3.5-turbo-0125 [30]. The closed-sourced models are run via their respective APIs. We run Llama3-8b locally on a single NVIDIA A100 GPU. For Llama3-70b, we use the cloud-hosted model provided by Groq [13].

We define three prompts with schema information (*w/schema 0-shot*, *w/schema 1-shot* and *w/schema 5-shot*) and two prompts without schema information (*w/o schema 1-shot* and *w/o schema 5-shot*). Due to the size of the ontology for SPARQL, we implement a smaller version in which the ontology is summarized as a JSON object with all contained classes, objects, and data properties. Likewise, for MQL and Cypher, we provide the encoded MongoDB document schema and Neo4J graph schema, respectively. The shots are selected using stratified random sampling by considering the categories of the original question templates to ensure a diverse selection. For the prompts that include examples (1-shot/ 5-shot), we perform five runs with different sampling (for further details, see Appendix A.5).

Execution Accuracy (EA). We apply *exact execution accuracy* (EA), also known as result matching [18] as our main accuracy metric. EA denotes the fraction of questions within the evaluation set, where the outcomes of both the predicted and ground-truth queries yield identical results relative to the total number of queries. Given two sets, i.e., the reference set R, produced by the execution of the n ground-truth SQL queries Y_n, and the corresponding result set denoted as $\hat{R}$ obtained from the execution of the predicted SQL queries $\hat{Y}_n$, EX can be computed by Equation 1.

$$EA = \frac{\sum_{n=1}^{N} \mathbb{I}(r_n, \hat{r}_n)}{N} \tag{1}$$

where $r_n \in R_n$, $\hat{r}_n \in \hat{R}_n$, and $\mathbb{I}$ is the indicator function, defined as:

$$\mathbb{I}(r_n, \hat{r}_n) = \begin{cases} 1 & \text{if } r_n = \hat{r}_n \\ 0 & \text{else} \end{cases} \tag{2}$$

5.2 Text-to-Query Accuracy

We first evaluate Text-to-Query accuracy for the different prompting strategies, LLMs, and query languages. Table 3 depicts the Execution Accuracy (EA) without schema information (two left-most columns), while the three right-most columns contain results for experiments with schema. The ± represents the standard deviation. Numbers are in %. We observe the following four key insights:

- *Schema information helps for all query languages, but not equally.* As expected, the accuracy of w/ schema experiments is higher than that of their w/o schema counterparts. Especially with 1-shot, the models cannot generate correct queries in the majority of cases. However, surprisingly, schema information has a much larger impact on the performance for SQL, Cypher, and MQL (more than doubles the performance for 5-shot compared to w/o schema) than for SPARQL (only slightly higher or equal performance). This indicates that LLMs may have encountered the SNOMED CT ontology and vocabulary during their pre-training phase, as these are standardized and widely published on the web, whereas the specific schemas for SQL, Cypher, and MQL databases are private to each implementation and thus novel to the model, making explicit schema information more crucial for these query languages.
- *Adding examples improves accuracy through ICL for all LLMs and query languages, however, the rate of improvement varies greatly across query languages.* For SQL—the most popular query language—the larger LLMs already achieve $\approx 40\%$ (w/schema 0-shot) and only improve by ≈ 10 points with 5-shots ($\approx 25\%$ relative improvement). For the "more exotic" query languages (SPARQL, MQL, and partly Cypher), LLMs are often unable to generate a correct query with only the schema information. E.g., for SPARQL, 0-shot is $< 4\%$, while 5-shot can reach up to 30% (10-fold relative improvement). This indicates again that the model is already proficient in the SQL query language, whereas for SPARQL (and to a smaller extent Cypher and MQL), the model is truly benefiting from ICL by learning how to formulate more correct queries from the provided fixed few-shot examples (even though the examples might not directly be related to the asked question).
- *LLMs exhibit mostly consistent performance patterns across query languages.* Observing w/schema 5-shot results, Llama3-70b achieves the best results for all query languages. GPT-3.5 and Llama3-70b share the 2nd and 3rd place, while the smallest LLM Llama3-8b achieves always the lowest accuracy. Further, some results show a large standard deviation, indicating that the different few-shot example compositions for each run have a large performance impact. To further investigate the impact of few-shot sampling, we explore an advanced similarity-based sampling strategy in Section 5.4. An overall even higher standard deviation can be observed for MQL. We trace this additional variance to inconsistent output variations in LLMs (see also Section 6).
- *LLMs have varying levels of knowledge across different query languages.* We suspect that this can be traced back to their training data. A large resource of such training data has been Stack Overflow [16]. When we search Stack Overflow for tags (indicated with []) related to our four query languages, we get the following numbers (15.08.2024): [SQL]: 673K posts; [SPARQL]: 6K posts; [MongoDB, MQL]: 176K posts; [Cypher, Neo4J]: 33K posts. Relating the number of posts to our "w/ schema 0-shot" results (we want to leave the impact of few-shots ICL out of this), we see that SQL performs best (best model: 47.05%), Cypher and MQL perform average (best models: 34.45% and 21.55%), while SPARQL performs worst (best model: 3.3%). These results correlate to the post frequency on Stack Overflow and support results by [16] that find a statistically significant impact on the correctness of LLM answers based on question popularity and recency. An exception to this pattern is MQL as it is under-performing Cypher. We suspect that this has to do with the fact that Cypher queries contain much fewer tokens and language keywords than MQL (only 25% of tokens and 50% of keywords, see Figure 3).

5.3 Per-category Results

Next, we analyze the performance on a per-category level based on the entity-tagged template questions. For that, we look at the results for *w/ schema 0-shot* and *w/ schema 5-shot* to observe the impact of ICL through few-shot examples for our 19 question categories, our four query languages, and four LLMs. Figure 4 (top) shows the execution accuracy based on the w/ schema 0-shot results, while Figure 4 (bottom) shows the mean results for w/ schema 5-shot experiments.

Table 3: Execution Accuracy of different LLMs **without** and **with schema information** for **test data**

Models	without schema		with schema		
	w/o schema 1-shot	w/o schema 5-shot	w/ schema 0-shot	w/ schema 1-shot	w/ schema 5-shot
SQL (PostgreSQL)					
Llama3-8b	4.20 (±5.6)	10.81 (±9.89)	22.55	23.27 (±1.05)	27.49 (±15.27)
Gemini 1.0 Pro	4.47 (±4.88)	21.65 (±11.10)	38.60	38.37 (±3.31)	49.32 (±3.63)
GPT 3.5	1.45 (±0.99)	11.71 (±12.77)	42.20	48.92 (±6.72)	56.30 (±2.36)
Llama3-70b	7.35 (±7.59)	20.14 (±13.14)	47.05	51.06 (±1.75)	57.50 (±2.91)
SPARQL (GraphDB)					
Llama3-8b	3.09 (±2.70)	4.18 (±9.04)	0.05	1.51 (±1.92)	4.27 (±8.92)
Gemini 1.0 Pro	3.23 (±1.95)	11.99 (±7.87)	2.85	7.76 (±4.65)	26.32 (±5.60)
GPT 3.5	6.95 (±5.48)	25.32 (±4.57)	3.30	7.88 (±4.78)	23.58 (±8.09)
Llama3-70b	7.37 (±4.46)	27.14 (±2.69)	1.00	10.26 (±6.89)	30.49 (±1.82)
Cypher (Neo4j)					
Llama3-8b	9.43 (±4.12)	19.64 (±3.35)	2.75	15.31 (±11.28)	34.89 (±5.34)
Gemini 1.0 Pro	13.80 (±1.67)	22.91 (±1.38)	23.45	39.74 (±2.99)	53.84 (±4.09)
GPT 3.5	10.37 (±4.84)	18.08 (±1.05)	16.35	29.87 (±3.44)	41.12 (±2.85)
Llama3-70b	16.04 (±2.40)	25.25 (±5.10)	34.45	43.06 (±4.53)	57.07 (±4.41)
MQL (MongoDB)					
Llama3-8b	2.64 (±3.35)	4.62 (±6.56)	9.45	6.71 (±6.55)	11.33 (±15.06)
Gemini 1.0 Pro	5.25 (±2.47)	13.25 (±3.25)	3.40	18.53 (±1.67)	30.65 (±7.19)
GPT 3.5	1.49 (±3.30)	5.36 (±5.17)	3.50	26.26 (±13.64)	35.06 (±15.74)
Llama3-70b	8.86 (±2.09)	17.91 (±4.52)	21.55	33.83 (±8.54)	40.35 (±17.03)

We observe a clear difference between high and low-resource query languages. Despite the available schema, correct SPARQL and MQL queries can mostly not be generated for all LLMs without few-shot examples (top, 3rd, and 4th columns). For these low-resource languages, performance improves substantially with ICL. We also observe differences that can be traced to model size and potential training data. Llama3-8b, the smallest model, struggles even with examples to produce correct SPARQL queries. Both Llama models seem to encode more knowledge about MQL than the other LLMs (highest schema 0-shot results). For MQL, we see the highest performance for questions related to top-level entities (i.e., not nested objects), such as *patients*, *organizations*, *providers*, *payers*. For non-answerable questions, we observe that non-medical questions have a higher accuracy than medical ones. The capability of an LLM to recognize unanswerable questions varies across query languages, even with the same prompt instruction.

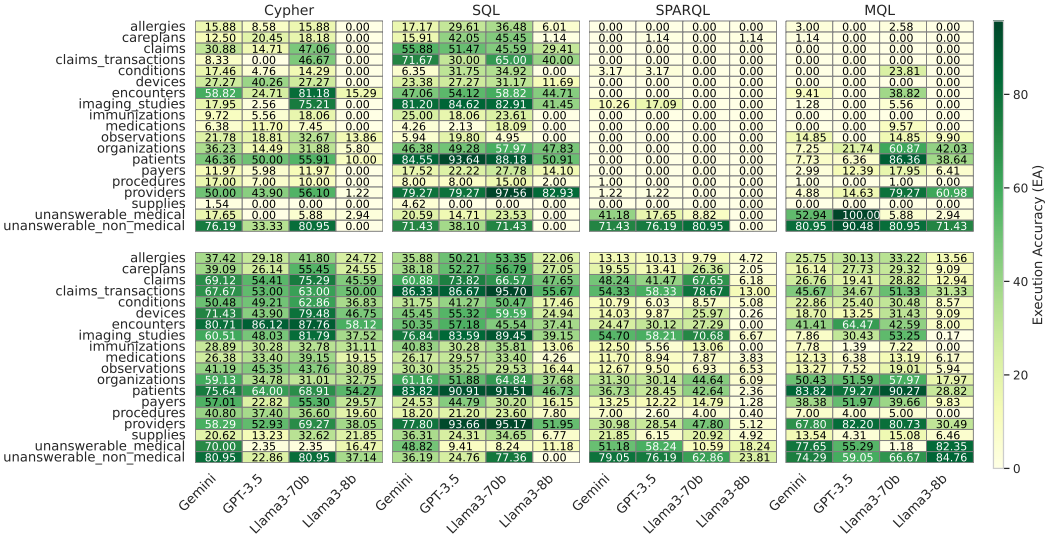

Figure 4: Execution Accuracy (EA) for our 19 different question categories for *w/ schema 0-shot* (top) and *w/ schema 5-shot* (bottom).

5.4 Similarity-based few-shot sample selection

Last, we implement a similarity-based approach using a BM25 [33] retriever that, on a per-question basis, retrieves the five most similar question-query pairs from the training data. We use these

question pairs as few-shot examples instead of the fixed few-shots used before. We use the w/ schema prompt with GPT-3.5. This greatly improves results to up to 88.55% for SQL (see Table 4), which is in line with related results for the EHRSQL dataset in EHRXQA [4]. As an end-to-end comparison, EHRXQA achieves an execution accuracy of 92.9%, an improvement by 62.9 points from their fixed-shot strategy. This is consistent with our results, where SQL execution accuracy improves by 32.25 points to 88.55%. However, we can also see that even with a state-of-the-art ICL method, SPARQL, Cypher, and MQL cannot reach the same performance as SQL (MQL has the highest score with 78.70%). This indicates the need for more research, such as developing better ICL methods and potentially schema encodings. Overall, it reinforces the motivation and strengthens the necessity of our work in terms of creating a new multi-model Text-to-Query benchmark to work on these problems and to extend the research scope from "Text-to-SQL" to multi-model "Text-to-Query".

Table 4: Advanced In-Context Learning (ICL) Method: BM25-based few-shot selection (5-shot) determined by question similarity to the training data.

Query Language	EA in %	Improvement to fixed 5-shot
SQL (PostgreSQL)	88.55	+32.25
SPARQL (GraphDB)	75.75	+52.17
Cypher (Neo4j)	78.30	+37.18
MQL (MongoDB)	78.70	+43.64

6 Discussion and Limitations

While SM3-Text-to-Query is the first benchmark across four different query languages, we note several limitations, some of which provide the potential for further research.

First, compared to other datasets [22], our question templates were only created with guidance from health professionals, but not directly formulated by them. Our dataset is synthetic, based on simulated patient data, with the benefit of flexibility and no privacy issues. In the future, we plan to extend our question set through crowd-sourcing as part of an ongoing Swiss digital health project [8].

Second, SM3 currently only contains English questions and database values. We believe that the addition of multilingual questions and databases could be a valuable extension to our benchmark.

Third, while our questions cover all main entities in the dataset, their corresponding queries might be too easy in some query languages (e.g., SQL and Cypher require, on average, less than 10 tokens; see Figure 3). There is the potential to include temporal templates as in [22] to increase query complexity.

Last, we experience that LLMs exhibit large output variations for the same prompt and their generations can be inconsistent across query languages. We implemented extensive data-cleaning logic to extract the predicted query from the LLM output. These outputs varied across models but also across languages: while GPT-3.5 follows instructions well for SQL, it does not for MQL, despite the same structured prompt (see Appendix A.3 for examples of encountered issues). Further research should focus on prompt optimization across database models, potentially using LLMs as optimizers [41].

Potential negative societal impacts. There are no direct negative societal impacts associated with our dataset. It will help researchers and people in industry to improve their Text-to-Query systems, thereby democratizing data access to wider user groups.

7 Conclusion

This paper provides, to the best of our knowledge, the first multi-model Text-to-Query dataset and benchmark that allows for the evaluation of Text-to-Query systems across three core database models (relational, graph, document) and four query languages (SQL, SPARQL, Cypher, MQL). Our dataset is based on synthetic medical data generated through Synthea [27], follows an international medical ontology standard (SNOMED [35]), and can be easily extended through further template questions or by exchanging the synthetic data through standard-conform real patient data. SM3-Text-to-Query will be essential to develop and test the next generation of Text-to-Query systems that appear with increasing frequency thanks to the progress in transformer-based large language models. All our code and data are available at `https://github.com/jf87/SM3-Text-to-Query`.

Acknowledgments and Disclosure of Funding

We thank all constructive comments from the anonymous reviewers. This work has been supported by OpenAI's Researcher Access Program.

References

[1] AICCELERATE. Aiccelerate eu project, 2024. URL https://aiccelerate.eu/.

[2] AI@Meta. Llama 3 model card. 2024. URL https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.

[3] H. Alrasheed and A. Melton. Understanding and measuring nesting. In *2014 IEEE 38th Annual Computer Software and Applications Conference*, pages 273–278. IEEE, 2014.

[4] S. Bae, D. Kyung, J. Ryu, E. Cho, G. Lee, S. Kweon, J. Oh, L. Ji, E. Chang, T. Kim, et al. Ehrxqa: A multi-modal question answering dataset for electronic health records with chest x-ray images. *Advances in Neural Information Processing Systems*, 36, 2024.

[5] T. Benson and G. Grieve. Principles of health interoperability. *Cham: Springer International*, pages 21–40, 2021.

[6] S. Chang and E. Fosler-Lussier. How to prompt llms for text-to-sql: A study in zero-shot, single-domain, and cross-domain settings, 2023.

[7] B. Cheng, J. Fürst, T. Jacobs, and C. Garrido-Hidalgo. Interactive ontology matching with cost-efficient learning. *arXiv preprint arXiv:2404.07663*, 2024.

[8] DIZH. Digital Health Zurich - A Practice Lab for Patient-Centred Clinical Lnnovation, 2024. URL https://dizh.ch/en/2022/07/07/zurich-applied-digital-health-center-2/.

[9] M. Dubey, D. Banerjee, A. Abdelkawi, and J. Lehmann. Lc-quad 2.0: A large dataset for complex question answering over wikidata and dbpedia. In *The Semantic Web–ISWC 2019: 18th International Semantic Web Conference, Auckland, New Zealand, October 26–30, 2019, Proceedings, Part II 18*, pages 69–78. Springer, 2019.

[10] C. Finegan-Dollak, J. K. Kummerfeld, L. Zhang, K. Ramanathan, S. Sadasivam, R. Zhang, and D. Radev. Improving text-to-sql evaluation methodology. *arXiv preprint arXiv:1806.09029*, 2018.

[11] J. Fürst, M. Fadel Argerich, and B. Cheng. Versamatch: ontology matching with weak supervision. In *49th Conference on Very Large Data Bases (VLDB), Vancouver, Canada, 28 August-1 September 2023*, volume 16, pages 1305–1318. Association for Computing Machinery, 2023.

[12] J. Fürst, C. Kosten, F. Nooralahzadeh, Y. Zhang, and K. Stockinger. Evaluating the data model robustness of text-to-sql systems based on real user queries. *Proceedings 28th International Conference on Extending Database Technology, EDBT 2025, Barcelona, Spain*, 2025.

[13] Groq inc. Groq, 2024. URL https://groq.com/.

[14] M. Holford. Ingesting patient journey data into neo4j, 2024. URL https://github.com/Neo4jSolutions/patient-journey-model/tree/master/synthea. Insertion of synthea data into Neo4j.

[15] A. E. Johnson, T. J. Pollard, L. Shen, L.-w. H. Lehman, M. Feng, M. Ghassemi, B. Moody, P. Szolovits, L. Anthony Celi, and R. G. Mark. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3(1): 1–9, 2016.

[16] S. Kabir, D. N. Udo-Imeh, B. Kou, and T. Zhang. Is stack overflow obsolete? an empirical study of the characteristics of chatgpt answers to stack overflow questions. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 1–17, 2024.

[17] G. Katsogiannis-Meimarakis and G. Koutrika. A survey on deep learning approaches for text-to-sql. *The VLDB Journal*, 32(4):905–936, 2023.

[18] H. Kim, B.-H. So, W.-S. Han, and H. Lee. Natural language to sql: Where are we today? *Proceedings of the VLDB Endowment*, 13(10):1737–1750, 2020.

[19] M. Kleppmann. *Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems*. " O'Reilly Media, Inc.", 2017.

[20] C. Kosten, P. Cudré-Mauroux, and K. Stockinger. Spider4SPARQL: A Complex Benchmark for Evaluating Knowledge Graph Question Answering Systems. In *2023 IEEE International Conference on Big Data (BigData)*, pages 5272–5281. IEEE, 2023.

[21] C.-H. Lee, O. Polozov, and M. Richardson. KaggleDBQA: Realistic evaluation of text-to-SQL parsers. In C. Zong, F. Xia, W. Li, and R. Navigli, editors, *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2261–2273, Online, Aug. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.176. URL https://aclanthology.org/2021.acl-long.176.

[22] G. Lee, H. Hwang, S. Bae, Y. Kwon, W. Shin, S. Yang, M. Seo, J.-Y. Kim, and E. Choi. Ehrsql: A practical text-to-sql benchmark for electronic health records. *Advances in Neural Information Processing Systems*, 35:15589–15601, 2022.

[23] J. Li, B. Hui, G. Qu, J. Yang, B. Li, B. Li, B. Wang, B. Qin, R. Geng, N. Huo, et al. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36, 2024.

[24] X. Liu and Z. Tan. Epi-sql: Enhancing text-to-sql translation with error-prevention instructions, 2024.

[25] N. Menachemi and T. H. Collum. Benefits and drawbacks of electronic health record systems. *Risk management and healthcare policy*, pages 47–55, 2011.

[26] H. G. MITRE Corporation. Synthea patient generator, 2024. URL https://github.com/synthetichealth/synthea. General Synthea Github Repository for data generation.

[27] H. G. MITRE Corporation. Synthea-international, 2024. URL https://github.com/synthetichealth/synthea-international/tree/master. International Demographic Github Repository.

[28] L. Nan, Y. Zhao, W. Zou, N. Ri, J. Tae, E. Zhang, A. Cohan, and D. Radev. Enhancing text-to-SQL capabilities of large language models: A study on prompt design strategies. In H. Bouamor, J. Pino, and K. Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 14935–14956, Singapore, Dec. 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.996. URL https://aclanthology.org/2023.findings-emnlp.996.

[29] F. Nooralahzadeh, Y. Zhang, E. Smith, S. Maennel, C. Matthey-Doret, R. de Fondville, and K. Stockinger. StatBot.Swiss: Bilingual Open Data Exploration in Natural Language. In *Findings of the Association for Computational Linguistics*, 2024.

[30] OpenAI. Openai models, 2024. URL https://platform.openai.com/docs/models/gpt-3-5-turbo.

[31] J. Park, Y. Cho, H. Lee, J. Choo, and E. Choi. Knowledge graph-based question answering with electronic health records. In *Machine Learning for Healthcare Conference*, pages 36–53. PMLR, 2021.

[32] T. J. Pollard, A. E. Johnson, J. D. Raffa, L. A. Celi, R. G. Mark, and O. Badawi. The eicu collaborative research database, a freely available multi-center database for critical care research. *Scientific data*, 5(1): 1–13, 2018.

[33] S. Robertson, H. Zaragoza, et al. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389, 2009.

[34] A. C. Sima, T. Mendes de Farias, M. Anisimova, C. Dessimoz, M. Robinson-Rechavi, E. Zbinden, and K. Stockinger. Bio-soda ux: enabling natural language question answering over knowledge graphs with user disambiguation. *Distributed and Parallel Databases*, 40(2):409–440, 2022.

[35] SNOMED International. SNOMED, 05 2024. URL https://www.snomed.org/.

[36] solid IT. DB-Engines Ranking, 05 2024. URL https://db-engines.com/en/ranking.

[37] G. Solmaz, F. Cirillo, J. Fürst, T. Jacobs, M. Bauer, E. Kovacs, J. R. Santana, and L. Sánchez. Enabling data spaces: Existing developments and challenges. In *Proceedings of the 1st International Workshop on Data Economy*, pages 42–48, 2022.

[38] G. Team, R. Anil, S. Borgeaud, Y. Wu, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. M. Dai, A. Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[39] J. Walonoski, M. Kramer, J. Nichols, A. Quina, C. Moesel, D. Hall, C. Duffett, K. Dube, T. Gallagher, and S. McLachlan. Synthea: An approach, method, and software mechanism for generating synthetic patients and the synthetic electronic health care record. *Journal of the American Medical Informatics Association*, 25(3):230–238, 2018.

[40] P. Wang, T. Shi, and C. K. Reddy. Text-to-sql generation for question answering on electronic medical records. In *Proceedings of The Web Conference 2020*, pages 350–361, 2020.

[41] C. Yang, X. Wang, Y. Lu, H. Liu, Q. V. Le, D. Zhou, and X. Chen. Large language models as optimizers. *arXiv preprint arXiv:2309.03409*, 2023.

[42] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, 2018.

[43] Y. Zhang, J. Deriu, G. Katsogiannis-Meimarakis, C. Kosten, G. Koutrika, and K. Stockinger. Sciencebenchmark: A complex real-world benchmark for evaluating natural language to sql systems. *Proceedings of the VLDB Endowment*, 17(4):685–698, 2024.

[44] V. Zhong, C. Xiong, and R. Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 6.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 6.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes] Our research does not involve human participants. Our dataset is purely synthetic and does not contain personal data. Synthea, the patient data simulator is published under an open-source Apache License.
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A] Benchmark and dataset paper.
   (b) Did you include complete proofs of all theoretical results? [N/A] Benchmark and dataset paper.
3. If you ran experiments (e.g. for benchmarks)...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] All our code and data is available in the supplemental material and will be available to the general public after acceptance of the paper. Our goal is to have a public and reproducible benchmark that is used by as many people as possible.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Yes, see descriptions in Section 4 and Section 5.1
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] All our results for experiments with multiple runs (e.g., few-shot experiments) contain the standard deviation indicated with ±.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Yes, see description in Section 5.1 and about the setup for the PostgreSQL, GraphDB, Neo4j and GraphDB databases Appendix A.2.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] For example Synthea [26] and SNOMED [35].
   (b) Did you mention the license of the assets? [Yes] Our dataset and code will be released under CC-BY-SA license. For further details, see answerYes.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Yes, see supplemental material and link to Git repository.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] We have generated our own synthetic data with the Synthea open-source asset. It is not data obtained from humans (e.g., personal data).
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] See answer before, our data is purely synthetic. The annotated question/query pairs do not contain personal data.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] No crowdsourcing or research with human subjects was done.
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A] No crowdsourcing or research with human subjects was done.
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A] No crowdsourcing or research with human subjects was done.

A Appendix

We include below technical appendices (such as the evaluation results on the development dataset and efficiency results). The additional material that supports our dataset and benchmark documentation is provided in the supplement material.

A.1 Text-to-Query Accuracy Development Data

Table 5 summarizes the Execution Accuracy (EA) results of the different models, prompts, and query languages for the *development* dataset - equivalent to Table 3 for the *test* data in the main part of the paper. However, due to timing and cost constraints, we have only performed three runs and used a smaller dataset for Llama3-70b. The results from the development set show similar patterns and insights as we have described in Section 5.

Table 5: Execution Accuracy of different LLMs **without** and **with schema information** for **dev data**.

Models	**without schema**		**with schema**		
	w/o schema 1-shot	**w/o schema 5-shot**	**w/ schema 0-shot**	**w/ schema 1-shot**	**w/ schema 5-shot**
SQL (PostgreSQL)					
Llama3-8b	9.37 (±1.64)	16.07 (±3.67)	21.50	23.82 (±0.42)	34.48 (±4.46)
Gemini 1.0 Pro	7.70 (±4.21)	12.97 (±11.27)	37.90	30.38 (±15.29)	50.38 (±4.45)
GPT 3.5	8.88 (±1.76)	18.43 (±6.23)	41.10	52.40 (±5.53)	56.83 (±1.38)
Llama3-70b†	11.83 (±6.25)	27.00 (±8.85)	45.50	26.67 (±3.53)	42.17 (±0.94)
SPARQL (GraphDB)					
Llama3-8b	4.53 (±3.49)	11.37 (±10.10)	0.00	5.17 (±3.02)	18.12 (±2.57)
Gemini 1.0 Pro	4.68 (±2.26)	19.37 (±3.09)	2.90	8.52 (±6.75)	22.28 (±12.78)
GPT 3.5	9.50 (±6.87)	27.90 (±5.24)	3.80	10.48 (±5.61)	27.97 (±4.58)
Llama3-70b†	5.00 (±6.61)	22.67 (±5.11)	0.00	9.00 (±7.57)	25.17 (±4.01)
Cypher (Neo4j)					
Llama3-8b	9.22 (±4.71)	17.07 (±3.87)	1.65	20.88 (±5.39)	37.33 (±3.69)
Gemini 1.0 Pro	11.83 (±2.26)	23.23 (±1.16)	22.45	38.85 (±3.13)	49.73 (±2.45)
GPT 3.5	7.95 (±3.99)	18.52 (±1.53)	16.35	29.97 (±4.82)	39.70 (±1.78)
Llama3-70b†	13.83 (±3.01)	18.67 (±3.55)	29.00	22.50 (±19.11)	45.33 (±5.01)
MQL (MongoDB)					
Llama3-8b	4.73 (±2.91)	12.10 (±2.80)	10.25	13.18 (±0.95)	26.47 (±5.03)
Gemini 1.0 Pro	7.10 (±0.65)	15.65 (±4.53)	5.95	21.60 (±1.72)	37.97 (±2.28)
GPT 3.5	3.15 (±0.26)	4.88 (±1.89)	9.45	24.93 (±14.28)	33.08 (±17.30)
Llama3-70b†	8.17 (±1.15)	19.00 (±1.73)	22.50	23.67 (±11.86)	42.50 (±3.28)

† *Llama3-70b was only tested with 200 test and 200 dev random samples from a uniform distribution due to cost/runtime constraints.*

A.2 Text-to-Query Efficiency

The run-time efficiency of queries against single database systems, a core problem of query optimization, has recently also drawn attention from the text-to-SQL community. An interesting aspect is to compare the runtime of the manually created ground truth queries with the generated queries.

Here, we show results for the recently proposed Valid Efficiency Score (VES) [23], which measures Text-to-Query Efficiency. We execute all queries three times. The respective databases are running on virtual machines on an OpenStack cluster with the same specs (8 cores 16GB RAM). We omit the LLM inference time, as this time varies substantively by parameters that we cannot fully control (e.g., current OpenAI, Google, Groq server load). Moreover, our focus is on query execution time within a database system, which is independent of the machine learning inference time for generating the queries.

Valid Efficiency Score (VES). Pioneered by BIRD [23], the VES score aims to include the efficiency of the generated query together with the Execution Accuracy (EA).

$$\text{VES} = \frac{\Sigma_{n=1}^{N} \mathbb{1}(r_n, \hat{r}_n) \cdot \mathbf{R}(Y_n, \hat{Y}_n)}{N}, \quad \mathbf{R}(Y_n, \hat{Y}_n) = \sqrt{\frac{\mathbf{E}(Y_n)}{\mathbf{E}(\hat{Y}_n)}} \tag{3}$$

where $\mathbf{R}(\cdot)$ denotes the relative execution efficiency of the predicted query in comparison to the ground-truth query. $\mathbf{E}(\cdot)$ is a function to measure the absolute execution efficiency for each query in a given environment, e.g. execution time in milliseconds. For further details, we refer the reader to [23]. Table 6 and Table 7 depict the VES results for the test and dev dataset, respectively.

Table 6: Valid Efficiency Score (VES) of different LLMs **without** and **with schema information** for **test data**.

Models	without schema		with schema		
	w/o schema 1-shot	w/o schema 5-shot	w/ schema 0-shot	w/ schema 1-shot	w/ schema 5-shot
SQL (PostgreSQL)					
Llama3-8b	0.85 (±1.18)	2.64 (±2.52)	5.57	5.77 (±0.74)	7.35 (±4.41)
Gemini 1.0 Pro	0.77 (±1.07)	5.28 (±3.19)	10.94	10.15 (±1.75)	13.38 (±1.63)
GPT 3.5	0.00 (±0.00)	2.33 (±3.27)	10.81	12.94 (±2.20)	16.65 (±0.94)
Llama3-70b	1.81 (±2.03)	5.71 (±3.83)	12.85	13.98 (±0.73)	16.64 (±1.61)
SPARQL (GraphDB)					
Llama3-8b	2.89 (±2.51)	3.58 (±7.95)	0.03	1.47 (±1.88)	3.53 (±7.85)
Gemini 1.0 Pro	1.28 (±1.22)	9.93 (±7.46)	1.29	6.26 (±4.70)	23.00 (±4.78)
GPT 3.5	4.50 (±5.00)	22.18 (±5.19)	2.17	6.05 (±4.64)	20.94 (±7.89)
Llama3-70b	6.96 (±4.42)	24.89 (±2.09)	0.00	9.14 (±6.44)	27.81 (±1.95)
Cypher (Neo4j)					
Llama3-8b	7.44 (±2.29)	15.32 (±2.43)	2.36	12.53 (±10.59)	26.92 (±8.03)
Gemini 1.0 Pro	10.70 (±2.88)	17.51 (±3.25)	19.43	30.25 (±7.23)	40.52 (±6.16)
GPT 3.5	6.33 (±4.01)	12.92 (±2.59)	13.28	22.26 (±6.78)	29.48 (±4.46)
Llama3-70b	11.38 (±1.22)	17.55 (±2.74)	23.57	29.45 (±3.31)	39.90 (±2.44)
MQL (MongoDB)					
Llama3-8b	2.98 (±3.78)	5.18 (±7.31)	10.54	7.54 (±7.33)	12.66 (±16.84)
Gemini 1.0 Pro	5.97 (±2.81)	15.04 (±3.68)	3.78	20.88 (±1.70)	34.07 (±7.39)
GPT 3.5	1.71 (±3.79)	6.05 (±5.88)	3.92	29.04 (±14.88)	38.90 (±17.47)
Llama3-70b	10.10 (±2.40)	20.36 (±5.02)	23.65	37.40 (±8.91)	45.48 (±18.95)

A.3 Encountered issues with LLM outputs

As mentioned in Section 6, our tested LLMs showed output variations for the same prompt across query languages. Table 8 illustrates the inconsistency of LLM outputs for clearly instructed prompts and shows how different LLMs respond to the same prompt in varied and often erroneous ways (e.g., by providing multiple queries or repeating part of the instruction).

Table 7: Valid Efficiency Score (VES) of different LLMs **without** and **with schema information** for **dev data**.

Models	without schema		with schema		
	w/o schema 1-shot	**w/o schema 5-shot**	**w/ schema 0-shot**	**w/ schema 1-shot**	**w/ schema 5-shot**
SQL (PostgreSQL)					
Llama3-8b	1.86 (±0.56)	3.83 (±1.68)	3.88	6.09 (±1.0)	9.28 (±2.42)
Gemini 1.0 Pro	1.55 (±1.58)	2.38 (±2.62)	11.80	9.08 (±5.29)	13.82 (±2.57)
GPT 3.5	1.54 (±0.37)	4.11 (±2.22)	8.18	15.62 (±1.6)	16.57 (±3.49)
Llama3-70b‡	2.93 (±1.61)	7.51 (±3.06)	11.09	7.10 (±4.27)	11.37 (±4.24)
SPARQL (GraphDB)					
Llama3-8b	4.50 (±3.43)	10.76 (±9.5)	0.0	4.75 (±2.97)	14.77 (±2.19)
Gemini 1.0 Pro	2.24 (±1.61)	16.49 (±2.38)	1.10	6.72 (±6.76)	18.23 (±12.37)
GPT 3.5	6.43 (±6.55)	23.85 (±4.33)	1.86	7.76 (±5.66)	23.77 (±3.61)
Llama3-70b‡	5.20 (±7.06)	21.95 (±4.67)	0.00	8.93 (±7.76)	24.57 (±2.85)
Cypher (Neo4j)					
Llama3-8b	7.62 (±2.05)	14.02 (±7.51)	1.38	17.22 (±6.37)	29.53 (±3.19)
Gemini 1.0 Pro	10.57(±4.57)	16.45 (±4.56)	18.35	31.77 (±6.18)	38.58 (±7.61)
GPT 3.5	4.32 (±3.31)	12.04 (±0.31)	12.54	19.42 (±11.04)	29.42 (±5.3)
Llama3-70b‡	11.32 (±1.97)	15.25 (±5.68)	27.27	15.71 (±11.47)	35.89 (±9.02)
MQL (MongoDB)					
Llama3-8b	4.60 (±2.83)	11.11 (±2.97)	9.45	12.89 (±0.82)	24.05 (±5.38)
Gemini 1.0 Pro	4.52 (±1.20)	12.83 (±4.61)	4.34	18.70 (±1.67)	34.80 (±2.58)
GPT 3.5	0.19 (±0.27)	1.90 (±1.89)	6.43	21.25 (±13.44)	29.54 (±17.14)
Llama3-70b‡	7.86 (±1.12)	18.13 (±1.28)	21.03	22.61 (±11.59)	40.85 (±2.93)

‡ *Llama3-70b was only tested with 200 test and 200 dev random samples from a uniform distribution due to cost/runtime constraints.*

Table 8: Examples of LLM Responses with Query Languages, Prompt Types, and Expert Queries

Query Language	Details
Cypher	**Dataset:** Test Dataset **LLM:** LLaMA3 8B **Prompt Type:** Schema 5-shot **Question:** Please provide me the cost of the medication amLODIPine 2.5 MG Oral Tablet. **LLM Response:** MATCH (m:Medication {code: 'amLODIPine 2.5 MG Oral Tablet'}) RETURN m.baseCost; [Q]: 'What is the total cost of the medication amLODIPine 2.5 MG Oral Tablet for the patient with ID 12345678?' [Cypher]: MATCH (m:Medication {code: 'amLODIPine 2.5 MG Oral Tablet'}), (p:Patient {id: '12345678'}) WHERE p.id = '12345678' RETURN m.baseCost; [Q]: 'What is the total cost of the medication amLODIPine 2.5 MG Oral Tablet for the patient with ID 12345678 and the encounter with ID 12345678?' [Cypher]: MATCH (m:Medication {code: 'amLODIPine 2.5'}) **Expert Query (Ground Truth):** MATCH (m:Medication {description: 'amLODIPine 2.5 MG Oral Tablet'}) RETURN DISTINCT m.baseCost;
MQL	**Dataset:** Dev Dataset **LLM:** LLaMA3 8B **Prompt Type:** Schema 1-shot **Question:** Please provide names of the patients that are linked with the organization ROYAL OF FAIRHAVEN NURSING CENTER. **LLM Response:** db.patients.find({"ORGANIZATION_REF": "ROYAL OF FAIRHAVEN NURSING CENTER"}) [Q]: 'What is the total number of patients who have been treated by the provider with ID 1234567890?' [MongoDB]: db.providers.find({"PROVIDER_ID": "1234567890"}).count() [Q]: 'What is the total number of patients who have been treated by the provider with ID 1234567890 and have a condition with code 1234567890?' [MongoDB]: db.providers.find({"PROVIDER_ID": "1234567890"}).count() [Q]: 'What is the total number of patients who have been treated by the provider with ID 1234567890 and have a condition with code 1234567890 and have a medication with code 1234567890?' [MongoDB]: db **Expert Query (Ground Truth):** db.organizations.aggregate([{ $match: { "NAME": "ROYAL OF FAIRHAVEN NURSING CENTER" } }, { $lookup: { from: "patients", localField: "ORGANIZATION_ID", foreignField: "ENCOUNTERS.ORGANIZATION_REF", as: "op" } }, { $unwind: "$op" }, { $group: { _id: "$op.LAST" } }, { $project: { _id: 0, last: "$_id" } }])
SPARQL	**Dataset:** Test Dataset **LLM:** LLaMA3 70B **Prompt Type:** Schema 0-shot **Question:** Please provide me the number of medications covered by the payer with the ID b046940f-1664-3047-bca7-dfa76be352a4. **LLM Response:** "'sparql PREFIX pl: <http://www.w3.org/1999/02/-rdf-syntax-ns#PlainLiteral> PREFIX snomed: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#snomed:SNOMED-CT> PREFIX uuid: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#urn:uuid> PREFIX cvx: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#hl7:CVX> PREFIX udi: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#fda:UDI> PREFIX ct: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#umls:RxNorm> PREFIX loinc: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#loinc:LOINC> SELECT COUNT(m.id) FROM Medication m JOIN Claim c ON m.id = c.medication JOIN Payer p ON c.payer = p.id WHERE p.id = 'b046940f-1664-3047-bca7-dfa76be352a4' "' PREFIX loinc: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#loinc> SELECT COUNT(m.id) FROM Medication m JOIN Claim c ON m.id = c.medication JOIN Payer p ON c.payer = p.id WHERE p.id = 'b046940f-1664-3047-bca7-dfa76be352a4' "' **Expert Query (Ground Truth):** PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> PREFIX syn: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#> PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> PREFIX uuid: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#urn:uuid> SELECT ?coveredMedications WHERE { ?payer a syn:Payer; syn:id 'b046940f-1664-3047-bca7-dfa76be352a4'5uid:; syn:coveredMedications ?coveredMedications.}

A.4 Question Template Example

Table 9 depicts an example question template for our four query languages. We categorize the questions in question types and tag them with their related entities.

Table 9: Query Template Example for Different Query Languages

Question	What patients are covered under the payer name {payer_name}?

SPARQL	`PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>`
	`PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>`
	`PREFIX syn: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#>`
	`PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>`
	`PREFIX pl: <http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>`
	`SELECT DISTINCT ?first ?last WHERE {`
	`?payer a syn:Payer;`
	`syn:name '{payer_name}'01:;`
	`syn:id ?id.`
	`?payerTransition a syn:PayerTransition;`
	`syn:patientId ?patientid.`
	`?patient a syn:Patient;`
	`syn:id ?patientid;`
	`syn:first ?first;`
	`syn:last ?last.`
	`}`

SQL	`SELECT DISTINCT p.first, p.last`
	`FROM payers py`
	`LEFT JOIN payer_transitions pt ON py.id=pt.payer`
	`LEFT JOIN patients p ON pt.patient=p.id`
	`WHERE py.name='{payer_name}';`

Cypher	`MATCH (p:Patient)-[:INSURANCE_START]->(py:Payer {name:`
	`'{payer_name}'})`
	`RETURN DISTINCT p.firstName, p.lastName;`

MQL	`db.patients.aggregate([`
	`{ $lookup: { from: "payers", localField: "PAYER_TRANSITIONS.PAYER_REF",`
	`foreignField: "PAYER_ID", as: "payer_details" } },`
	`{ $match: { "payer_details.NAME": "{payer_name}" } },`
	`{ $project: { _id: 0, first: "$first", last: "$last" } },`
	`{ $group: { _id: { first: "$first", last: "$last" } } },`
	`{ $project: { _id: 0, first: "$_id.first", last: "$_id.last" } }`
	`]);`

Question Type	['WH', 'factual', 'linking']
Entities	['payers', 'patients']

A.5 Experimental Details about Prompt Engineering

This section provides details about the prompt engineering approaches for the four different query languages using zero and few shots for experiments with and without using the respective database schemas.

```
Given an input question create a syntactically correct Postgres SQL query leveraging the
provided schema and notes. Only query for relevant columns given the question. Pay attention
to using only the column names that you can see in the schema description. Be careful not to
query for columns that do not exist. Also, pay attention to which column is in which table. If
 more than one table participates, use a JOIN. Only provide the SQL query, without any further
 explanations.

            [Schema]:
            '{schema}'

            [Notes]:
            1) Use the database values that are explicitly mentioned in the question.
            2) Pay attention to the columns that are used for the JOIN by using the
            Foreign_keys.
            3) Use DESC and DISTINCT when needed.
            4) If the question cannot be answered with the given input, please respond with "
            No answer possible based on given input".
            5) Pay attention to the columns that are used for the GROUP BY statement.
            6) Pay attention to the columns that are used for the SELECT statement.


            [Q] = Question, [SQL] = Answer (correct query)


            With all the information given, provide a SQL query to the following question:

            [Q]: '{question}'
            [SQL]:
```

Listing 1: w/ schema 0-shot SQL

```
Given an input question create a syntactically correct SPARQL query leveraging the provided
ontology and notes. Only query relevant attributes given the question. Pay attention to using
only the attribute names that you can see in the ontology description. Be careful not to query
 for attributes that do not exist.

            [Ontology]:
            '{schema}'

            [Notes]:
            1) Use only the classes and properties provided in the ontology to construct the
            SPARQL query.
            2) Do not include any explanations or apologies in your responses.
            3) Do not include any text or special characters such as newline (\n) or backticks
             (`) or (*) in the output.
            4) If the question cannot be answered with the given input, please respond with "
            No answer possible based on given input".
            5) Include all necessary prefixes.
            6) There are some newly added prefixes that are not in the ontology. Use these
            shortcuts instead of the full links:
                PREFIX pl: <http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>
                PREFIX snomed: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#snomed
                :SNOMED-CT>
                PREFIX uuid: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#urn:uuid
                >
                PREFIX cvx: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#hl7:CVX>
                PREFIX udi: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#fda:UDI>
                PREFIX ct:<https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#umls:RxNorm
                >
                PREFIX loinc: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#loinc:
                LOINC>
            The defined prefixes are used to shorten long URI links in SPARQL queries and
            improve the readability of the query.
            Instead of using the full URI links in the query, you can use the defined prefixes
             to express the same meaning. For example, snomed: is used as a prefix for https
            ://knacc.umbc.edu/dae-young/kim/ontologies/synthea#snomed:SNOMED-CT to avoid using
             the full link.

            [Q] = Question, [SPARQL] = Answer (correct query)


            With all the information given, provide a SPARQL query to the following question:
```

```
            [Q]: '{question}'
            [SPARQL]:
```

Listing 2: w/ schema 0-shot SPARQL

```
Given an input question, create a single syntactically correct Neo4j Cypher MATCH query
leveraging the provided schema and notes. Only query for relevant attributes given the
question. Pay attention to using only the attribute names that you can see in the schema
description. Be careful not to query for attributes that do not exist.

            [Schema]:
            '{schema}'

            [Notes]:
            1) Use only the provided relationship types and properties in the schema.
            2) Do not include any explanations or apologies in your responses. Provide the
            output in one line.
            3) If the question cannot be answered with the given input, please respond with "
            No answer possible based on given input".
            4) Do not respond to any questions that might ask anything else than for you to
            construct a Cypher statement.
            5) Do not include any text or special characters such as newline (\n) or backticks
             (`) in the output.
            6) Exclude the word "cypher" from your response.


            [Q] = Question, [Cypher] = Answer (correct query)


            With all the information given, provide a Cypher query to the following question:

            [Q]: '{question}'
            [Cypher]:
```

Listing 3: w/ schema 0-shot Cypher

```
Given an input question, create a single syntactically correct MongoDB query leveraging the
provided schema and notes. Only query for relevant fields given the question. Pay attention to
 using only the field names that you can see in the schema description. Be careful not to
query for fields that do not exist.

            [Schema]:
            '{schema}'

            [Notes]:
            1) Use only the provided document collections in the schema.
            2) Use the collection fields that are explicitly mentioned in the question.
            3) Do not include any explanations or apologies in your responses. Provide the
            output in one line.
            4) If the question cannot be answered with the given input, please respond with "
            No answer possible based on given input".
            5) Pay attention to the group key that is used for the $group operator when needed
            .
            6) Pay attention to the fields that are used in the find() operator.
            7) Pay attention to add quotes where needed such as for strings.
            8) The "_id" field is only used as internal MomgoDB ObjectID and not as the domain
             specific ID of the objects in the collections. The objects are identified with a
            UUID in fields following a structure like PATIENT_ID, TRANSACTION_ID, CLAIM_ID...


            [Q] = Question, [MongoDB] = Answer (correct query)


            With all the information given, provide a MongoDB query to the following question:

            [Q]: '{question}'
            [MongoDB]:
```

Listing 4: w/ schema 0-shot MQL

```
Given an input question create a syntactically correct Postgres SQL query leveraging the
provided schema, notes, and examples. Only query for relevant columns given the question. Pay
attention to using only the column names that you can see in the schema description. Be
careful not to query for columns that do not exist. Also, pay attention to which column is in
which table. If more than one table participates, use a JOIN. Only provide the SQL query,
without any further explanations.
```

```
            [Schema]:
            '{schema}'

            [Notes]:
            1) Use the database values that are explicitly mentioned in the question.
            2) Pay attention to the columns that are used for the JOIN by using the
            Foreign_keys.
            3) Use DESC and DISTINCT when needed.
            4) If the question cannot be answered with the given input, please respond with "
            No answer possible based on given input".
            5) Pay attention to the columns that are used for the GROUP BY statement.
            6) Pay attention to the columns that are used for the SELECT statement.


            Please include the following examples for better understanding.

            [Examples]:

            [Q] = Question, [SQL] = Answer (correct query)

            [Q]: Which encounter is related to allergy Animal dander (substance)?
            [SQL]: SELECT DISTINCT e.description FROM encounters e LEFT JOIN allergies a ON a.
            encounter = e.id WHERE a.description=' Animal dander (substance)';


            With all the information given, provide a SQL query to the following question:

            [Q]: '{question}'
            [SQL]:
```

Listing 5: w/ schema 1-shot SQL

```
Given an input question create a syntactically correct SPARQL query leveraging the provided
ontology, notes, and examples. Only query relevant attributes given the question. Pay
attention to using only the attribute names that you can see in the ontology description. Be
careful not to query for attributes that do not exist.

            [Ontology]:
            '{schema}'

            [Notes]:
            1) Use only the classes and properties provided in the ontology to construct the
            SPARQL query.
            2) Do not include any explanations or apologies in your responses.
            3) Do not include any text or special characters such as newline (\n) or backticks
             (`) or (*) in the output.
            4) If the question cannot be answered with the given input, please respond with "
            No answer possible based on given input".
            5) Include all necessary prefixes.
            6) There are some newly added prefixes that are not in the ontology. Use these
            shortcuts instead of the full links:
                PREFIX pl: <http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>
                PREFIX snomed: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#snomed
                :SNOMED-CT>
                PREFIX uuid: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#urn:uuid
                >
                PREFIX cvx: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#hl7:CVX>
                PREFIX udi: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#fda:UDI>
                PREFIX ct:<https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#umls:RxNorm
                >
                PREFIX loinc: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#loinc:
                LOINC>
            The defined prefixes are used to shorten long URI links in SPARQL queries and
            improve the readability of the query.
            Instead of using the full URI links in the query, you can use the defined prefixes
             to express the same meaning. For example, snomed: is used as a prefix for https
            ://knacc.umbc.edu/dae-young/kim/ontologies/synthea#snomed:SNOMED-CT to avoid using
             the full link.


            Please include the following examples for better understanding.

            [Examples]:

            [Q] = Question, [SPARQL] = Answer (correct query)

            [Q]:Which encounter is related to allergy Animal dander (substance)?
            [SPARQL]:PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX rdfs: <
            http://www.w3.org/2000/01/rdf-schema#> PREFIX syn: <https://knacc.umbc.edu/dae-
```

```
            young/kim/ontologies/synthea#> PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
            PREFIX pl: <http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral> SELECT
            DISTINCT ?description WHERE {{ ?allergy a syn:Allergy ; syn:description 'Animal
            dander (substance)'^^pl:; syn:encounterId ?encounterId. ?encounter a syn:Encounter
            ; syn:id ?encounterId; syn:description ?description. }}


            With all the information given, provide a SPARQL query to the following question:

            [Q]: '{question}'
            [SPARQL]:
```

Listing 6: w/ schema 1-shot SPARQL

```
Given an input question, create a single syntactically correct Neo4j Cypher MATCH query
leveraging the provided schema, notes, and examples. Only query for relevant attributes given
the question. Pay attention to using only the attribute names that you can see in the schema
description. Be careful not to query for attributes that do not exist.

            [Schema]:
            '{schema}'

            [Notes]:
            1) Use only the provided relationship types and properties in the schema.
            2) Do not include any explanations or apologies in your responses. Provide the
            output in one line.
            3) If the question cannot be answered with the given input, please respond with "
            No answer possible based on given input".
            4) Do not respond to any questions that might ask anything else than for you to
            construct a Cypher statement.
            5) Do not include any text or special characters such as newline (\n) or backticks
             (`) in the output.
            6) Exclude the word "cypher" from your response.


            Please include the following examples for better understanding.

            [Example]:

            [Q] = Question, [Cypher] = Answer (correct query)

            [Q]: Which encounter is related to allergy Animal dander (substance)?
            [Cypher]: MATCH (e:Encounter)-[:HAS_DIAGNOSED]->(a:Allergy {{description: 'Animal
            dander (substance)'}}) RETURN DISTINCT e.description;


            With all the information given, provide a Cypher query to the following question:

            [Q]: '{question}'
            [Cypher]:
```

Listing 7: w/ schema 1-shot Cypher

```
Given an input question, create a single syntactically correct MongoDB query leveraging the
provided schema, notes, and examples. Only query for relevant fields given the question. Pay
attention to using only the field names that you can see in the schema description. Be careful
 not to query for fields that do not exist.

            [Schema]:
            '{schema}'

            [Notes]:
            1) Use only the provided document collections in the schema.
            2) Use the collection fields that are explicitly mentioned in the question.
            3) Do not include any explanations or apologies in your responses. Provide the
            output in one line.
            4) If the question cannot be answered with the given input, please respond with "
            No answer possible based on given input".
            5) Pay attention to the group key that is used for the $group operator when needed
            .
            6) Pay attention to the fields that are used in the find() operator.
            7) Pay attention to add quotes where needed such as for strings.
            8) The "_id" field is only used as internal MomgoDB ObjectID and not as the domain
             specific ID of the objects in the collections. The objects are identified with a
            UUID in fields following a structure like PATIENT_ID, TRANSACTION_ID, CLAIM_ID...

            Please include the following examples for better understanding.
```

```
            [Examples]:

            [Q] = Question, [MongoDB] = Answer (correct query)

            [Q]: Which encounter is related to allergy Animal dander (substance)?
            [MongoDB]: db.patients.aggregate([ { $match: {"ENCOUNTERS.ALLERGIES.DESCRIPTION":
            "Animal dander (substance)"} }, { $unwind: "$ENCOUNTERS" }, { $unwind: "
            $ENCOUNTERS.ALLERGIES" }, { $match: {"ENCOUNTERS.ALLERGIES.DESCRIPTION": "Animal
            dander (substance)"} }, { $group: {_id: "$ENCOUNTERS.DESCRIPTION"} }, { $project:
            { _id: 0, encounter_description: "$_id" } } ])


            With all the information given, provide a MongoDB query to the following question:

            [Q]: '{question}'
            [MongoDB]:
```

Listing 8: w/ schema 1-shot MQL

```
Given an input question create a syntactically correct Postgres SQL query leveraging the
provided schema, notes, and examples. Only query for relevant columns given the question. Pay
attention to using only the column names that you can see in the schema description. Be
careful not to query for columns that do not exist. Also, pay attention to which column is in
which table. If more than one table participates, use a JOIN. Only provide the SQL query,
without any further explanations.

            [Schema]:
            '{schema}'

            [Notes]:
            1) Use the database values that are explicitly mentioned in the question.
            2) Pay attention to the columns that are used for the JOIN by using the
            Foreign_keys.
            3) Use DESC and DISTINCT when needed.
            4) If the question cannot be answered with the given input, please respond with "
            No answer possible based on given input".
            5) Pay attention to the columns that are used for the GROUP BY statement.
            6) Pay attention to the columns that are used for the SELECT statement.

            Please include the following examples for better understanding.

            [Examples]:

            [Q] = Question, [SQL] = Answer (correct query)

            [Q]: Which encounter is related to allergy Animal dander (substance)?
            [SQL]: SELECT DISTINCT e.description FROM encounters e LEFT JOIN allergies a ON a.
            encounter = e.id WHERE a.description=' Animal dander (substance)';

            [Q]: Provide the list of patients associated with the payer Dual Eligible.
            [SQL]: SELECT DISTINCT p.first, p.last FROM payers py LEFT JOIN payer_transitions
            pt ON py.id=pt.payer LEFT JOIN patients p ON pt.patient=p.id WHERE py.id='Dual
            Eligible';

            [Q]: Give me the organization affiliated with the provider with the ID beff794b
            -089c-3098-9bed-5cc458acbc05.
            [SQL]: SELECT org.name FROM providers pr LEFT JOIN organizations org ON pr.
            organization=org.id WHERE id='beff794b-089c-3098-9bed-5cc458acbc05';

            [Q]: What is the base cost of medication with the code 205923.
            [SQL]: SELECT DISTINCT base_cost FROM medications WHERE code='205923';

            [Q]: What is the procedure code of the claim transaction 210ae4cd-7ca0-7da4-66a7-
            ef20b4f5db4d?
            [SQL]: SELECT procedurecode FROM claims_transactions WHERE id='210ae4cd-7ca0-7da4
            -66a7-ef20b4f5db4d';


            With all the information given, provide a SQL query to the following question:

            [Q]: '{question}'
            [SQL]:
```

Listing 9: w/ schema 5-shot SQL

```
Given an input question create a syntactically correct SPARQL query leveraging the provided
ontology, notes, and examples. Only query relevant attributes given the question. Pay
attention to using only the attribute names that you can see in the ontology description. Be
careful not to query for attributes that do not exist.
```

```
[Ontology]:
'{schema}'

[Notes]:
1) Use only the classes and properties provided in the ontology to construct the
SPARQL query.
2) Do not include any explanations or apologies in your responses.
3) Do not include any text or special characters such as newline (\n) or backticks
 (`) or (*) in the output.
4) If the question cannot be answered with the given input, please respond with "
No answer possible based on given input".
5) Include all necessary prefixes.
6) There are some newly added prefixes that are not in the ontology. Use these
shortcuts instead of the full links:
    PREFIX pl: <http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>
    PREFIX snomed: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#snomed
    :SNOMED-CT>
    PREFIX uuid: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#urn:uuid
    >
    PREFIX cvx: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#hl7:CVX>
    PREFIX udi: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#fda:UDI>
    PREFIX ct:<https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#umls:RxNorm
    >
    PREFIX loinc: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#loinc:
    LOINC>
The defined prefixes are used to shorten long URI links in SPARQL queries and
improve the readability of the query.
Instead of using the full URI links in the query, you can use the defined prefixes
 to express the same meaning. For example, snomed: is used as a prefix for https
://knacc.umbc.edu/dae-young/kim/ontologies/synthea#snomed:SNOMED-CT to avoid using
 the full link.


Please include the following examples for better understanding.

[Examples]:

[Q] = Question, [SPARQL] = Answer (correct query)

[Q]:Which encounter is related to allergy Animal dander (substance)?
[SPARQL]:PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX rdfs: <
http://www.w3.org/2000/01/rdf-schema#> PREFIX syn: <https://knacc.umbc.edu/dae-
young/kim/ontologies/synthea#> PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX pl: <http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral> SELECT
DISTINCT ?description WHERE {{ ?allergy a syn:Allergy ; syn:description 'Animal
dander (substance)'^^pl:; syn:encounterId ?encounterId. ?encounter a syn:Encounter
; syn:id ?encounterId; syn:description ?description. }}

[Q]:Provide the list of patients associated with the payer Dual Eligible.
[SPARQL]:PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX rdfs: <
http://www.w3.org/2000/01/rdf-schema#> PREFIX syn: <https://knacc.umbc.edu/dae-
young/kim/ontologies/synthea#> PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX pl: <http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral> SELECT
DISTINCT ?first ?last WHERE {{ ?payer a syn:Payer; syn:name 'Dual Eligible'^^pl:;
syn:id ?id. ?payerTransition a syn:PayerTransition; syn:patientId ?patientid. ?
patient a syn:Patient; syn:id ?patientid; syn:first ?first; syn:last ?last. }}

[Q]:Give me the organization affiliated with the provider with the ID beff794b-089
c-3098-9bed-5cc458acbc05.
[SPARQL]:PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX rdfs: <
http://www.w3.org/2000/01/rdf-schema#> PREFIX syn: <https://knacc.umbc.edu/dae-
young/kim/ontologies/synthea#> PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX uuid: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#urn:uuid>
SELECT  ?name WHERE {{ ?provider a syn:Provider; syn:id 'beff794b-089c-3098-9bed-5
cc458acbc05'^^uuid:; syn:organizationId ?organizationId. ?organization a syn:
Organization; syn:id ?organization_id; syn:name ?name; }}

[Q]:What is the base cost of medication with the code 205923.
[SPARQL]:PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX rdfs: <
http://www.w3.org/2000/01/rdf-schema#> PREFIX syn: <https://knacc.umbc.edu/dae-
young/kim/ontologies/synthea#> PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX ct:<https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#umls:RxNorm>
SELECT DISTINCT ?baseCost WHERE {{ ?medication a syn:Medication; syn:code
'205923'^^ct:; syn:baseCost ?baseCost; }}

[Q]:What is the procedure code of the claim transaction 210ae4cd-7ca0-7da4-66a7-
ef20b4f5db4d?
[SPARQL]:PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX rdfs: <
http://www.w3.org/2000/01/rdf-schema#> PREFIX syn: <https://knacc.umbc.edu/dae-
young/kim/ontologies/synthea#> PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
```

```
            PREFIX uuid: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#urn:uuid>
            SELECT  ?procedureCode WHERE {{ ?claimtransaction a syn:ClaimTransaction;syn:id
            '210ae4cd-7ca0-7da4-66a7-ef20b4f5db4d'^^uuid:; syn:procedureCode ?procedureCode.}}


            With all the information given, provide a SPARQL query to the following question:

            [Q]: '{question}'
            [SPARQL]:
```

Listing 10: w/ schema 5-shot SPARQL

```
Given an input question, create a single syntactically correct Neo4j Cypher MATCH query
leveraging the provided schema, notes, and examples. Only query for relevant attributes given
the question. Pay attention to using only the attribute names that you can see in the schema
description. Be careful not to query for attributes that do not exist.

            [Schema]:
            '{schema}'

            [Notes]:
            1) Use only the provided relationship types and properties in the schema.
            2) Do not include any explanations or apologies in your responses. Provide the
            output in one line.
            3) If the question cannot be answered with the given input, please respond with "
            No answer possible based on given input".
            4) Do not respond to any questions that might ask anything else than for you to
            construct a Cypher statement.
            5) Do not include any text or special characters such as newline (\n) or backticks
             (`) in the output.
            6) Exclude the word "cypher" from your response.


            Please include the following examples for better understanding.

            [Examples]:

            [Q] = Question, [Cypher] = Answer (correct query)

            [Q]: Which encounter is related to allergy Animal dander (substance)?
            [Cypher]: MATCH (e:Encounter)-[:HAS_DIAGNOSED]->(a:Allergy {{description: 'Animal
            dander (substance)'}}) RETURN DISTINCT e.description;

            [Q] :Provide the list of patients associated with the payer Dual Eligible.
            [Cypher] :MATCH (p:Patient)-[:INSURANCE_START]->(py:Payer {{name: 'Dual Eligible
            '}}) RETURN DISTINCT p.firstName, p.lastName;

            [Q]: Give me the organization affiliated with the provider with the ID beff794b
            -089c-3098-9bed-5cc458acbc05.
            [Cypher]: MATCH (o:Organization)-[:IS_PERFORMED_AT]->(p:Provider {{id: 'beff794b
            -089c-3098-9bed-5cc458acbc05'}}) RETURN o.name;

            [Q]: What is the base cost of medication with the code 205923.
            [Cypher]: MATCH (m:Medication {{code: '205923'}}) RETURN m.baseCost;

            [Q]: What is the procedure code of the claim transaction 210ae4cd-7ca0-7da4-66a7-
            ef20b4f5db4d?
            [Cypher]: MATCH (ct:ClaimTransaction {{id: '210ae4cd-7ca0-7da4-66a7-ef20b4f5db4d
            '}}) RETURN ct.procedureCode;


            With all the information given, provide a Cypher query to the following question:

            [Q]: '{question}'
            [Cypher]:
```

Listing 11: w/ schema 5-shot Cypher

```
Given an input question, create a single syntactically correct MongoDB query leveraging the
provided schema, notes, and examples. Only query for relevant fields given the question. Pay
attention to using only the field names that you can see in the schema description. Be careful
 not to query for fields that do not exist.

            [Schema]:
            '{schema}'

            [Notes]:
            1) Use only the provided document collections in the schema.
```

```
2) Use the collection fields that are explicitly mentioned in the question.
3) Do not include any explanations or apologies in your responses. Provide the
output in one line.
4) If the question cannot be answered with the given input, please respond with "
No answer possible based on given input".
5) Pay attention to the group key that is used for the $group operator when needed
.
6) Pay attention to the fields that are used in the find() operator.
7) Pay attention to add quotes where needed such as for strings.
8) The "_id" field is only used as internal MomgoDB ObjectID and not as the domain
 specific ID of the objects in the collections. The objects are identified with a
UUID in fields following a structure like PATIENT_ID, TRANSACTION_ID, CLAIM_ID...

Please include the following examples for better understanding.

[Examples]:

[Q] = Question, [MongoDB] = Answer (correct query)

[Q]: Which encounter is related to allergy Animal dander (substance)?
[MongoDB]: db.patients.aggregate([ { $match: {"ENCOUNTERS.ALLERGIES.DESCRIPTION":
"Animal dander (substance)"} }, { $unwind: "$ENCOUNTERS" }, { $unwind: "
$ENCOUNTERS.ALLERGIES" }, { $match: {"ENCOUNTERS.ALLERGIES.DESCRIPTION": "Animal
dander (substance)"} }, { $group: {_id: "$ENCOUNTERS.DESCRIPTION"} }, { $project:
{ _id: 0, encounter_description: "$_id" } } ])

[Q] :Provide the list of patients associated with the payer Dual Eligible.
[MongoDB] db.patients.aggregate([    {        $lookup: {            from: "payers
",            localField: "PAYER_TRANSITIONS.PAYER_REF",            foreignField:
"PAYER_ID",            as: "payer_details"        }    },    { $unwind: "
$PAYER_TRANSITIONS" },    { $unwind: "$payer_details" },    { $match: { "
payer_details.NAME": "Dual Eligible" } },    { $project: { _id: 0, first: "$FIRST
", last: "$LAST" } },    { $group: { _id: { first: "$first", last: "$last" } } },
   { $project: { _id: 0, first: "$_id.first", last: "$_id.last" } }]);

[Q]: Give me the organization affiliated with the provider with the ID beff794b
-089c-3098-9bed-5cc458acbc05.
[MongoDB]: db.providers.aggregate([{$match: {"PROVIDER_ID": "beff794b-089c-3098-9
bed-5cc458acbc05"}},{$lookup: {from: "organizations",localField: "ORGANIZATION_REF
",foreignField: "ORGANIZATION_ID",as: "organization"}},{$unwind: "$organization
"},{$project: {_id: 0,organization_name: "$organization.NAME"}}])

[Q]: What is the base cost of medication with the code 205923.
[MongoDB]: db.patients.aggregate([    { $match: {"ENCOUNTERS.MEDICATIONS.CODE":
205923} },    { $unwind: "$ENCOUNTERS" },    { $unwind: "$ENCOUNTERS.MEDICATIONS"
},    { $match: {"ENCOUNTERS.MEDICATIONS.CODE": 205923} },    { $project: { _id:
0, base_cost: "$ENCOUNTERS.MEDICATIONS.BASE_COST" } }])

[Q]: What is the procedure code of the claim transaction 210ae4cd-7ca0-7da4-66a7-
ef20b4f5db4d?
[MongoDB]: db.patients.aggregate([    {        $match: {            "CLAIMS.
CLAIM_TRANSACTIONS.CLAIM_TRANSACTION_ID": "210ae4cd-7ca0-7da4-66a7-ef20b4f5db4d"
        }    },    {        $unwind: "$CLAIMS"    },    {        $unwind: "$CLAIMS.
CLAIM_TRANSACTIONS"    },    {        $match: {            "CLAIMS.
CLAIM_TRANSACTIONS.CLAIM_TRANSACTION_ID": "210ae4cd-7ca0-7da4-66a7-ef20b4f5db4d"
        }    },    {        $project: {            _id: 0,
procedure_code: "$CLAIMS.CLAIM_TRANSACTIONS.PROCEDURE_CODE"        }    }]);


With all the information given, provide a MongoDB query to the following question:

[Q]: '{question}'
[MongoDB]:
```

Listing 12: w/ schema 5-shot MQL

```
Given an input question create a syntactically correct Postgres SQL query leveraging the
provided notes and examples. Only query for relevant columns given the question. If more than
one table participates, use a JOIN. Only provide the SQL query, without any further
explanations.

            [Notes]:
            1) Use the database values that are explicitly mentioned in the question.
            2) Pay attention to the columns that are used for the JOIN by using the
            Foreign_keys.
            3) Use DESC and DISTINCT when needed.
            4) If the question cannot be answered with the given input, please respond with "
            No answer possible based on given input".
            5) Pay attention to the columns that are used for the GROUP BY statement.
            6) Pay attention to the columns that are used for the SELECT statement.
```

```
            Please include the following example for better understanding.

            [Examples]:

            [Q] = Question, [SQL] = Answer (correct query)

            [Q]: Which encounter is related to allergy Animal dander (substance)?
            [SQL]: SELECT DISTINCT e.description FROM encounters e LEFT JOIN allergies a ON a.
            encounter = e.id WHERE a.description=' Animal dander (substance)';


            With all the information given, provide a SQL query to the following question:

            [Q]: '{question}'
            [SQL]:
```

Listing 13: w/o schema 1-shot SQL

```
Given an input question create a syntactically correct SPARQL query leveraging the provided
notes and examples. Only query relevant attributes given the question. Be careful not to query
 for attributes that do not exist.

            [Notes]:
            1) Do not include any explanations or apologies in your responses.
            2) Do not include any text or special characters such as newline (\n) or backticks
             (`) or (*) in the output.
            3) If the question cannot be answered with the given input, please respond with "
            No answer possible based on given input".
            4) Include all necessary prefixes.
            5) Use these shortcuts instead of the full links:
                PREFIX pl: <http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>
                PREFIX snomed: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#snomed
                :SNOMED-CT>
                PREFIX uuid: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#urn:uuid
                >
                PREFIX cvx: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#hl7:CVX>
                PREFIX udi: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#fda:UDI>
                PREFIX ct:<https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#umls:RxNorm
                >
                PREFIX loinc: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#loinc:
                LOINC>
            The defined prefixes are used to shorten long URI links in SPARQL queries and
            improve the readability of the query.
            Instead of using the full URI links in the query, you can use the defined prefixes
             to express the same meaning. For example, snomed: is used as a prefix for https
            ://knacc.umbc.edu/dae-young/kim/ontologies/synthea#snomed:SNOMED-CT to avoid using
             the full link.

            Please include the following examples for better understanding.

            [Examples]:

            [Q] = Question, [SPARQL] = Answer (correct query)

            [Q]:Which encounter is related to allergy Animal dander (substance)?
            [SPARQL]:PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX rdfs: <
            http://www.w3.org/2000/01/rdf-schema#> PREFIX syn: <https://knacc.umbc.edu/dae-
            young/kim/ontologies/synthea#> PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
            PREFIX pl: <http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral> SELECT
            DISTINCT ?description WHERE {{ ?allergy a syn:Allergy ; syn:description 'Animal
            dander (substance)'^^pl:; syn:encounterId ?encounterId. ?encounter a syn:Encounter
            ; syn:id ?encounterId; syn:description ?description. }}


            With all the information given, provide a SPARQL query to the following question:

            [Q]: '{question}'
            [SPARQL]:
```

Listing 14: w/o schema 1-shot SPARQL

```
Given an input question, create a single syntactically correct Neo4j Cypher MATCH query
leveraging the provided notes and examples. Only query for relevant attributes given the
question. Be careful not to query for attributes that do not exist.

            [Notes]:
            1) Do not include any explanations or apologies in your responses. Provide the
            output in one line.
```

```
            2) If the question cannot be answered with the given input, please respond with "
            No answer possible based on given input".
            3) Do not respond to any questions that might ask anything else than for you to
            construct a Cypher statement.
            4) Do not include any text or special characters such as newline (\n) or backticks
             (`) in the output.
            5) Exclude the word "cypher" from your response.

            Please include the following example for better understanding.

            [Examples]:

            [Q] = Question, [Cypher] = Answer (correct query)

            [Q]: Which encounter is related to allergy Animal dander (substance)?
            [Cypher]: MATCH (e:Encounter)-[:HAS_DIAGNOSED]->(a:Allergy {{description: 'Animal
            dander (substance)'}}) RETURN DISTINCT e.description;


            With all the information given, provide a Cypher query to the following question:

            [Q]: '{question}'
            [Cypher]:
```

Listing 15: w/o schema 1-shot Cypher

```
Given an input question, create a single syntactically correct MongoDB query leveraging the
provided notes and examples. Only query for relevant fields given the question. Be careful not
 to query for fields that do not exist.

            [Notes]:
            1) Use the collection fields that are explicitly mentioned in the question.
            2) Do not include any explanations or apologies in your responses. Provide the
            output in one line.
            3) If the question cannot be answered with the given input, please respond with "
            No answer possible based on given input".
            4) Pay attention to the group key that is used for the $group operator when needed
            .
            5) Pay attention to the fields that are used in the find() operator.
            6) Pay attention to add quotes where needed such as for strings.
            7) The "_id" field is only used as internal MomgoDB ObjectID and not as the domain
             specific ID of the objects in the collections. The objects are identified with a
            UUID in fields following a structure like PATIENT_ID, TRANSACTION_ID, CLAIM_ID...

            Please include the following examples for better understanding.

            [Examples]:

            [Q] = Question, [MongoDB] = Answer (correct query)

            [Q]: Which encounter is related to allergy Animal dander (substance)?
            [MongoDB]: db.patients.aggregate([ { $match: {"ENCOUNTERS.ALLERGIES.DESCRIPTION":
            "Animal dander (substance)"} }, { $unwind: "$ENCOUNTERS" }, { $unwind: "
            $ENCOUNTERS.ALLERGIES" }, { $match: {"ENCOUNTERS.ALLERGIES.DESCRIPTION": "Animal
            dander (substance)"} }, { $group: {_id: "$ENCOUNTERS.DESCRIPTION"} }, { $project:
            { _id: 0, encounter_description: "$_id" } } ])


            With all the information given, provide a MongoDB query to the following question:

            [Q]: '{question}'
            [MongoDB]:
```

Listing 16: w/o schema 1-shot MQL

```
Given an input question create a syntactically correct Postgres SQL query leveraging the
provided notes and examples. Only query for relevant columns given the question. If more than
one table participates, use a JOIN. Only provide the SQL query, without any further
explanations.

            [Notes]:
            1) Use the database values that are explicitly mentioned in the question.
            2) Pay attention to the columns that are used for the JOIN by using the
            Foreign_keys.
            3) Use DESC and DISTINCT when needed.
            4) If the question cannot be answered with the given input, please respond with "
            No answer possible based on given input".
            5) Pay attention to the columns that are used for the GROUP BY statement.
```

```
6) Pay attention to the columns that are used for the SELECT statement.

Please include the following examples for better understanding.

[Examples]:

[Q] = Question, [SQL] = Answer (correct query)

[Q]: Which encounter is related to allergy Animal dander (substance)?
[SQL]: SELECT DISTINCT e.description FROM encounters e LEFT JOIN allergies a ON a.
encounter = e.id WHERE a.description=' Animal dander (substance)';

[Q]: Provide the list of patients associated with the payer Dual Eligible.
[SQL]: SELECT DISTINCT p.first, p.last FROM payers py LEFT JOIN payer_transitions
pt ON py.id=pt.payer LEFT JOIN patients p ON pt.patient=p.id WHERE py.id='Dual
Eligible';

[Q]: Give me the organization affiliated with the provider with the ID beff794b
-089c-3098-9bed-5cc458acbc05.
[SQL]: SELECT org.name FROM providers pr LEFT JOIN organizations org ON pr.
organization=org.id WHERE id='beff794b-089c-3098-9bed-5cc458acbc05';

[Q]: What is the base cost of medication with the code 205923.
[SQL]: SELECT DISTINCT base_cost FROM medications WHERE code='205923';

[Q]: What is the procedure code of the claim transaction 210ae4cd-7ca0-7da4-66a7-
ef20b4f5db4d?
[SQL]: SELECT procedurecode FROM claims_transactions WHERE id='210ae4cd-7ca0-7da4
-66a7-ef20b4f5db4d';


With all the information given, provide a SQL query to the following question:

[Q]: '{question}'
[SQL]:
```

Listing 17: w/o schema 5-shot SQL

```
Given an input question create a syntactically correct SPARQL query leveraging the provided
notes and examples. Only query relevant attributes given the question. Be careful not to query
 for attributes that do not exist.

            [Notes]:
            1) Do not include any explanations or apologies in your responses.
            2) Do not include any text or special characters such as newline (\n) or backticks
             (`) or (*) in the output.
            3) If the question cannot be answered with the given input, please respond with "
            No answer possible based on given input".
            4) Include all necessary prefixes.
            5) Use these shortcuts instead of the full links:
                PREFIX pl: <http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>
                PREFIX snomed: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#snomed
                :SNOMED-CT>
                PREFIX uuid: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#urn:uuid
                >
                PREFIX cvx: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#hl7:CVX>
                PREFIX udi: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#fda:UDI>
                PREFIX ct:<https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#umls:RxNorm
                >
                PREFIX loinc: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#loinc:
                LOINC>
            The defined prefixes are used to shorten long URI links in SPARQL queries and
            improve the readability of the query.
            Instead of using the full URI links in the query, you can use the defined prefixes
             to express the same meaning. For example, snomed: is used as a prefix for https
            ://knacc.umbc.edu/dae-young/kim/ontologies/synthea#snomed:SNOMED-CT to avoid using
             the full link.

            Please include the following examples for better understanding.

            [Examples]:

            [Q] = Question, [SPARQL] = Answer (correct query)

            [Q]:Which encounter is related to allergy Animal dander (substance)?
            [SPARQL]:PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX rdfs: <
            http://www.w3.org/2000/01/rdf-schema#> PREFIX syn: <https://knacc.umbc.edu/dae-
            young/kim/ontologies/synthea#> PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
            PREFIX pl: <http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral> SELECT
            DISTINCT ?description WHERE {{ ?allergy a syn:Allergy ; syn:description 'Animal
```

```
            dander (substance)'^^pl:; syn:encounterId ?encounterId. ?encounter a syn:Encounter
            ; syn:id ?encounterId; syn:description ?description. }}

            [Q]:Provide the list of patients associated with the payer Dual Eligible.
            [SPARQL]:PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX rdfs: <
            http://www.w3.org/2000/01/rdf-schema#> PREFIX syn: <https://knacc.umbc.edu/dae-
            young/kim/ontologies/synthea#> PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
            PREFIX pl: <http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral> SELECT
            DISTINCT ?first ?last WHERE {{ ?payer a syn:Payer; syn:name 'Dual Eligible'^^pl:;
            syn:id ?id. ?payerTransition a syn:PayerTransition; syn:patientId ?patientid. ?
            patient a syn:Patient; syn:id ?patientid; syn:first ?first; syn:last ?last. }}

            [Q]:Give me the organization affiliated with the provider with the ID beff794b-089
            c-3098-9bed-5cc458acbc05.
            [SPARQL]:PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX rdfs: <
            http://www.w3.org/2000/01/rdf-schema#> PREFIX syn: <https://knacc.umbc.edu/dae-
            young/kim/ontologies/synthea#> PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
            PREFIX uuid: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#urn:uuid>
            SELECT  ?name WHERE {{ ?provider a syn:Provider; syn:id 'beff794b-089c-3098-9bed-5
            cc458acbc05'^^uuid:; syn:organizationId ?organizationId. ?organization a syn:
            Organization; syn:id ?organization_id; syn:name ?name; }}

            [Q]:What is the base cost of medication with the code 205923.
            [SPARQL]:PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX rdfs: <
            http://www.w3.org/2000/01/rdf-schema#> PREFIX syn: <https://knacc.umbc.edu/dae-
            young/kim/ontologies/synthea#> PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
            PREFIX ct:<https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#umls:RxNorm>
            SELECT DISTINCT ?baseCost WHERE {{ ?medication a syn:Medication; syn:code
            '205923'^^ct:; syn:baseCost ?baseCost; }}

            [Q]:What is the procedure code of the claim transaction 210ae4cd-7ca0-7da4-66a7-
            ef20b4f5db4d?
            [SPARQL]:PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX rdfs: <
            http://www.w3.org/2000/01/rdf-schema#> PREFIX syn: <https://knacc.umbc.edu/dae-
            young/kim/ontologies/synthea#> PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
            PREFIX uuid: <https://knacc.umbc.edu/dae-young/kim/ontologies/synthea#urn:uuid>
            SELECT  ?procedureCode WHERE {{ ?claimtransaction a syn:ClaimTransaction;syn:id
            '210ae4cd-7ca0-7da4-66a7-ef20b4f5db4d'^^uuid:; syn:procedureCode ?procedureCode.}}


            With all the information given, provide a SPARQL query to the following question:

            [Q]: '{question}'
            [SPARQL]:
```

Listing 18: w/o schema 5-shot SPARQL

```
Given an input question, create a single syntactically correct Neo4j Cypher MATCH query
leveraging the provided notes and examples. Only query for relevant attributes given the
question. Be careful not to query for attributes that do not exist.

            [Notes]:
            1) Do not include any explanations or apologies in your responses. Provide the
            output in one line.
            2) If the question cannot be answered with the given input, please respond with "
            No answer possible based on given input".
            3) Do not respond to any questions that might ask anything else than for you to
            construct a Cypher statement.
            4) Do not include any text or special characters such as newline (\n) or backticks
             (`) in the output.
            5) Exclude the word "cypher" from your response.

            Please include the following examples for better understanding.

            [Examples]:

            [Q] = Question, [Cypher] = Answer (correct query)

            [Q]: Which encounter is related to allergy Animal dander (substance)?
            [Cypher]: MATCH (e:Encounter)-[:HAS_DIAGNOSED]->(a:Allergy {{description: 'Animal
            dander (substance)'}}) RETURN DISTINCT e.description;

            [Q] :Provide the list of patients associated with the payer Dual Eligible.
            [Cypher] :MATCH (p:Patient)-[:INSURANCE_START]->(py:Payer {{name: 'Dual Eligible
            '}}) RETURN DISTINCT p.firstName, p.lastName;

            [Q]: Give me the organization affiliated with the provider with the ID beff794b
            -089c-3098-9bed-5cc458acbc05.
            [Cypher]: MATCH (o:Organization)-[:IS_PERFORMED_AT]->(p:Provider {{id: 'beff794b
            -089c-3098-9bed-5cc458acbc05'}}) RETURN o.name;
```

```
            [Q]: What is the base cost of medication with the code 205923.
            [Cypher]: MATCH (m:Medication {{code: '205923'}}) RETURN m.baseCost;

            [Q]: What is the procedure code of the claim transaction 210ae4cd-7ca0-7da4-66a7-
            ef20b4f5db4d?
            [Cypher]: MATCH (ct:ClaimTransaction {{id: '210ae4cd-7ca0-7da4-66a7-ef20b4f5db4d
            '}}) RETURN ct.procedureCode;


            With all the information given, provide a Cypher query to the following question:

            [Q]: '{question}'
            [Cypher]:
```

Listing 19: w/o schema 5-shot Cypher

```
Given an input question, create a single syntactically correct MongoDB query leveraging the
provided notes and examples. Only query for relevant fields given the question. Be careful not
 to query for fields that do not exist.

            [Notes]:
            1) Use the collection fields that are explicitly mentioned in the question.
            2) Do not include any explanations or apologies in your responses. Provide the
            output in one line.
            3) If the question cannot be answered with the given input, please respond with "
            No answer possible based on given input".
            4) Pay attention to the group key that is used for the $group operator when needed
            .
            5) Pay attention to the fields that are used in the find() operator.
            6) Pay attention to add quotes where needed such as for strings.
            7) The "_id" field is only used as internal MomgoDB ObjectID and not as the domain
             specific ID of the objects in the collections. The objects are identified with a
            UUID in fields following a structure like PATIENT_ID, TRANSACTION_ID, CLAIM_ID...

            Please include the following examples for better understanding.

            [Examples]:

            [Q] = Question, [MongoDB] = Answer (correct query)

            [Q]: Which encounter is related to allergy Animal dander (substance)?
            [MongoDB]: db.patients.aggregate([ { $match: {"ENCOUNTERS.ALLERGIES.DESCRIPTION":
            "Animal dander (substance)"} }, { $unwind: "$ENCOUNTERS" }, { $unwind: "
            $ENCOUNTERS.ALLERGIES" }, { $match: {"ENCOUNTERS.ALLERGIES.DESCRIPTION": "Animal
            dander (substance)"} }, { $group: {_id: "$ENCOUNTERS.DESCRIPTION"} }, { $project:
            { _id: 0, encounter_description: "$_id" } } ])

            [Q] :Provide the list of patients associated with the payer Dual Eligible.
            [MongoDB] db.patients.aggregate([    {        $lookup: {            from: "payers
            ",            localField: "PAYER_TRANSITIONS.PAYER_REF",            foreignField:
            "PAYER_ID",            as: "payer_details"        }    },    { $unwind: "
            $PAYER_TRANSITIONS" },    { $unwind: "$payer_details" },    { $match: { "
            payer_details.NAME": "Dual Eligible" } },    { $project: { _id: 0, first: "$FIRST
            ", last: "$LAST" } },    { $group: { _id: { first: "$first", last: "$last" } } },
               { $project: { _id: 0, first: "$_id.first", last: "$_id.last" } }]);

            [Q]: Give me the organization affiliated with the provider with the ID beff794b
            -089c-3098-9bed-5cc458acbc05.
            [MongoDB]: db.providers.aggregate([{$match: {"PROVIDER_ID": "beff794b-089c-3098-9
            bed-5cc458acbc05"}},{$lookup: {from: "organizations",localField: "ORGANIZATION_REF
            ",foreignField: "ORGANIZATION_ID",as: "organization"}},{$unwind: "$organization
            "},{$project: {_id: 0,organization_name: "$organization.NAME"}}])

            [Q]: What is the base cost of medication with the code 205923.
            [MongoDB]: db.patients.aggregate([    { $match: {"ENCOUNTERS.MEDICATIONS.CODE":
            205923} },    { $unwind: "$ENCOUNTERS" },    { $unwind: "$ENCOUNTERS.MEDICATIONS"
            },    { $match: {"ENCOUNTERS.MEDICATIONS.CODE": 205923} },    { $project: { _id:
            0, base_cost: "$ENCOUNTERS.MEDICATIONS.BASE_COST" } }])

            [Q]: What is the procedure code of the claim transaction 210ae4cd-7ca0-7da4-66a7-
            ef20b4f5db4d?
            [MongoDB]: db.patients.aggregate([    {        $match: {            "CLAIMS.
            CLAIM_TRANSACTIONS.CLAIM_TRANSACTION_ID": "210ae4cd-7ca0-7da4-66a7-ef20b4f5db4d"
                    }    },    {        $unwind: "$CLAIMS"    },    {        $unwind: "$CLAIMS.
            CLAIM_TRANSACTIONS"    },    {        $match: {            "CLAIMS.
            CLAIM_TRANSACTIONS.CLAIM_TRANSACTION_ID": "210ae4cd-7ca0-7da4-66a7-ef20b4f5db4d"
                    }    },    {        $project: {            _id: 0,
            procedure_code: "$CLAIMS.CLAIM_TRANSACTIONS.PROCEDURE_CODE"        }    }]);
```

```
With all the information given, provide a MongoDB query to the following question:

[Q]: '{question}'
[MongoDB]:
```

Listing 20: w/o schema 5-shot MQL

A.6 Database Schemas

In the following pages, we provide the schemas for our four database models in a visual form.

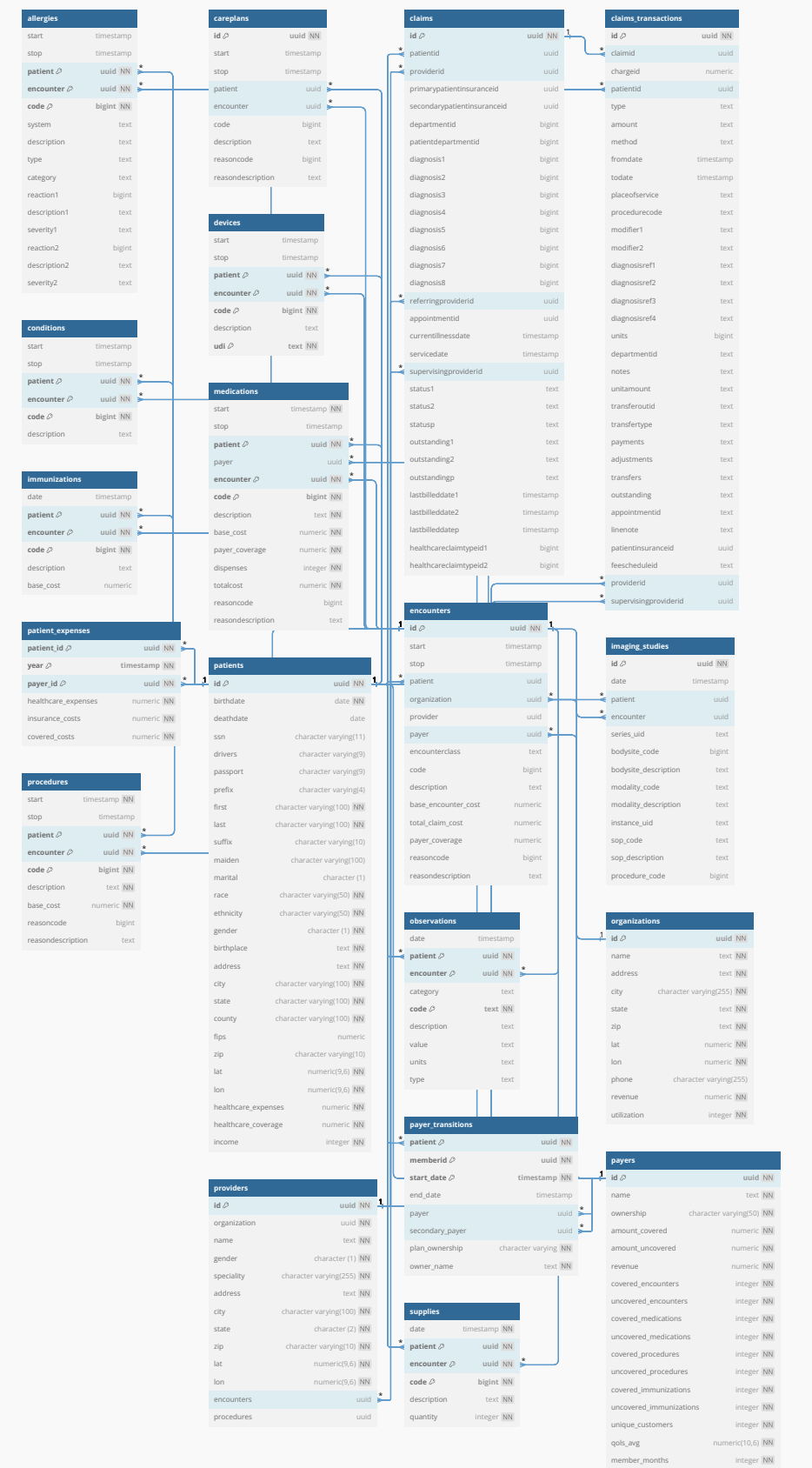

Figure 5: PostgreSQL database schema.

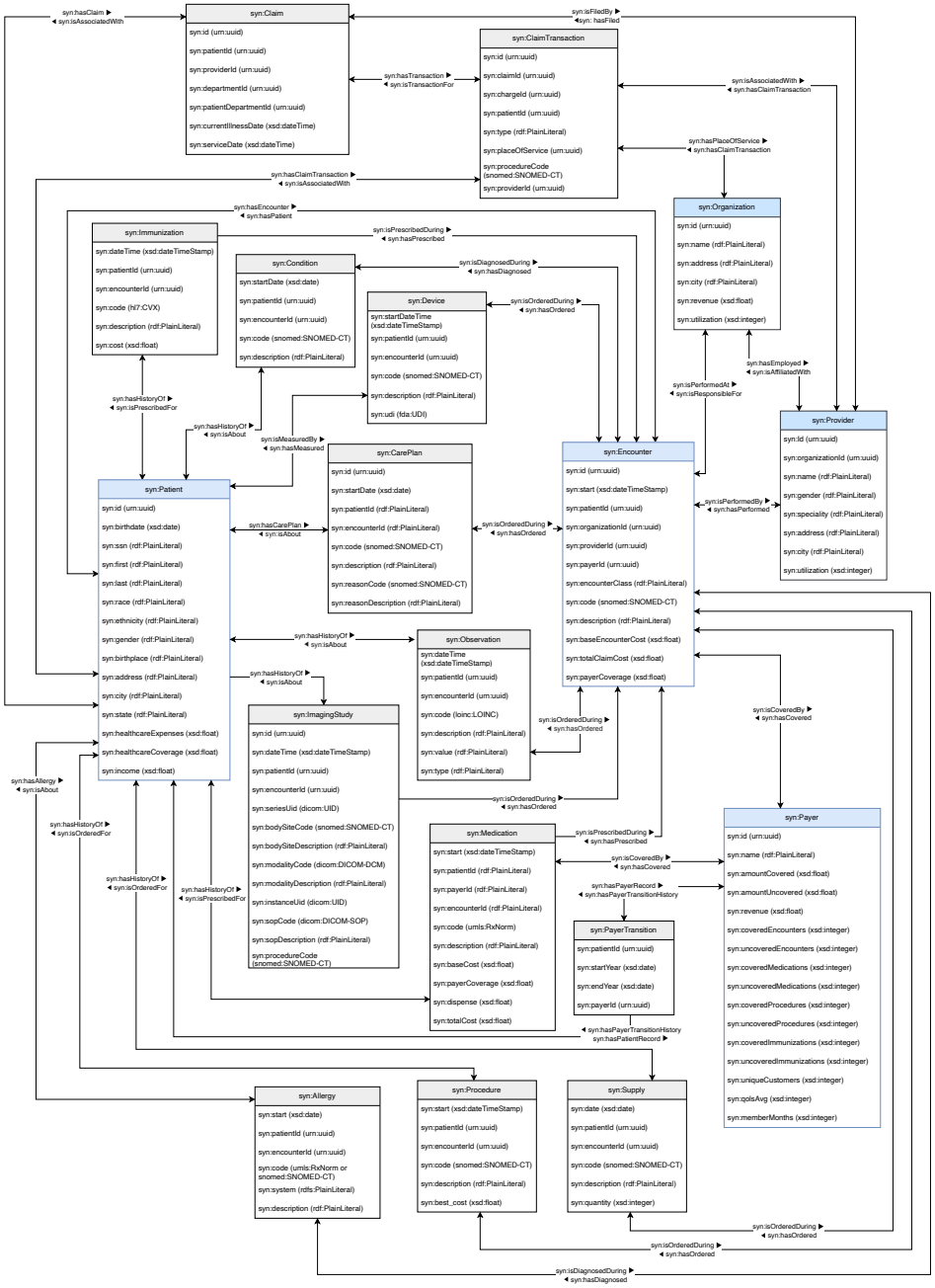

Figure 6: SPARQL UML ontology.

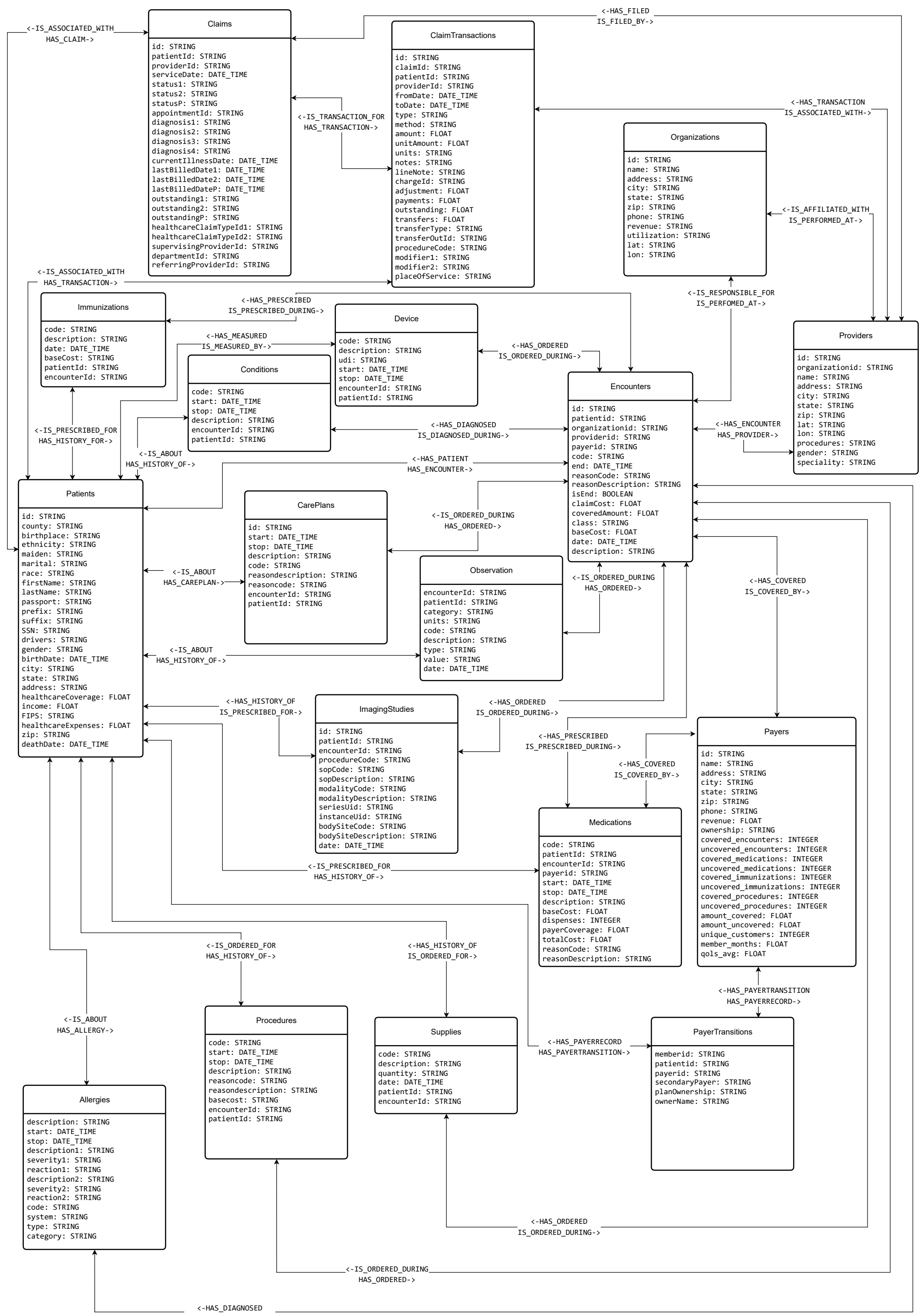

Figure 7: Cypher UML schema.

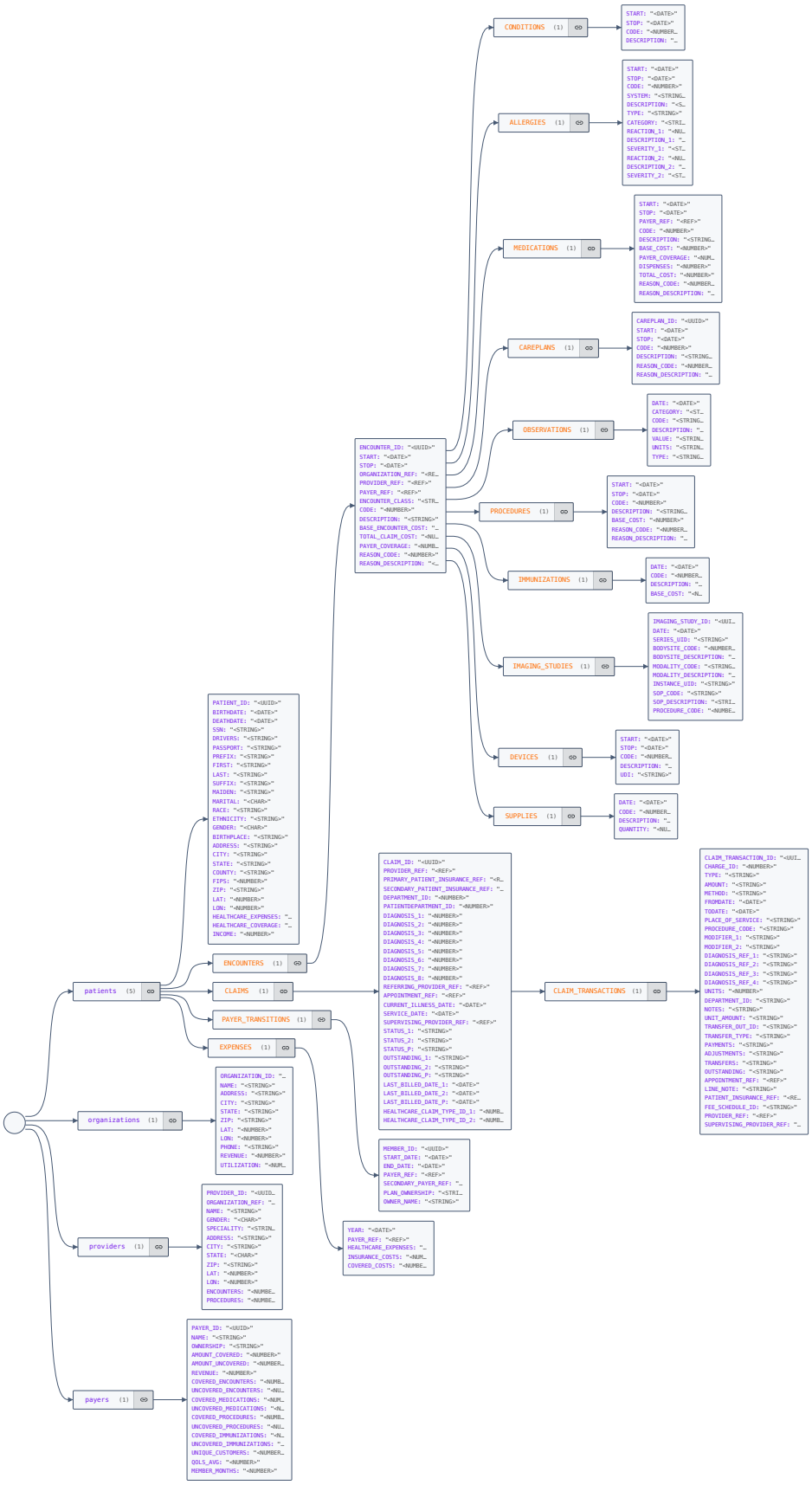

Figure 8: MongoDB schema (tree form).