

CHAPTER 1

Overview

If we are to satisfy the needs of casual users of data bases, we must break through the barriers that presently prevent these users from freely employing their native languages. – Ted Codd

After introducing the relational model for database management, which earned him the Turing Award, Tedd Codd worked on a project called Rendezvous [Codd \[1974\]](#), a dialog-based system he envisioned back in 1974, aimed at allowing casual users to effectively communicate with structured databases. The above quote was also made in the context of this project. However, at the time, there was a total lack of resources for this project, with the field of natural language processing at its infancy and the scarcity of digitized text and computing resources. Huge progress has been made in the past 50 years, especially in the area of natural language processing and a significant reduction in the cost of computing resources. This has pushed Codd's vision of Rendezvous to the forefront of current research and development.

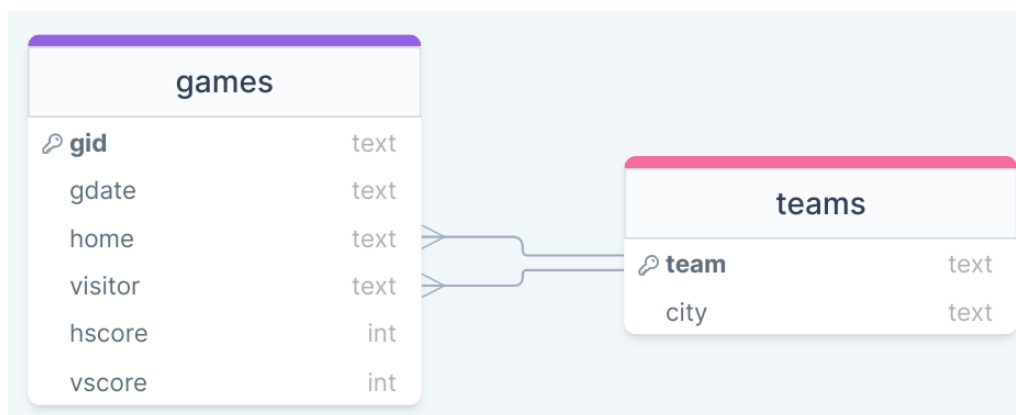


Figure 1.1: An example database

1.1 A SESSION WITH CHATGPT

ChatGPT is a chatbot developed by OpenAI and is built on top of a family of Large Language Models (LLMs) including GPT-3. The chatbot mimics human conversations using a set of large language models, each trained on some specific tasks. The use of those language models enables the chatbot to write essays, answer questions, compose music

2 1. OVERVIEW

and write code. ChatGPT remembers previous prompts within the same conversation, making the conversation more natural.

Consider a database with schema shown in Figure 1.1. The database has a table called *teams* that records the team names and their cities, and a table called *games* that records the games between different teams. Each game has a home team and a visitor team, and for each game both the home score and the visitor score, denoted by *hscore* and *vscore* respectively, are recorded.

–Tables

```
create table teams (  
  team text,  
  city text,  
  primary key(team)  
);  
create table games (  
  gid text,  
  gdate date,  
  home text,  
  visitor text,  
  hscore int,  
  vscore int,  
  primary key (gid),  
  foreign key (home) references teams,  
  foreign key (visitor) references teams  
);
```

– Queries

Write a SQLite query to find teams that have played a game against Oilers.

```
SELECT DISTINCT visitor  
FROM games  
WHERE home='Oilers'  
UNION  
SELECT DISTINCT home  
FROM games  
WHERE visitor='Oilers';
```

Table 1.1: Our session with ChatGPT (the model produced response is in red)

Suppose we want to find out all teams that have played a game against *Oilers*, and we want to express the query in SQL. Having ChatGPT at our disposal, we give the table definitions and the query as a prompt to ChatGPT. Table 1.1 shows the prompt that is given and the chatbot response shown in red. Without giving any additional information, the chatbot figures out that *hscore* refers to the score of the home team and *vscore* refers to the score of the visitor team, and it generates a correct SQL query. Our prompt

ends the first table definition by a colon instead of a semicolon, but the chatbot simply ignores the typo.

Write a SQLite query to find for each team that had a home game, the number of wins as a home team.

`SELECT home, COUNT(CASE WHEN hscore > vscore THEN 1 END) AS wins
FROM games
GROUP BY home;`

Write a SQLite query to teams that had more home wins than home losses.

`SELECT home, COUNT(CASE WHEN hscore > vscore THEN 1 END) AS wins,
COUNT(CASE WHEN hscore < vscore THEN 1 END) AS losses
FROM games
GROUP BY home
HAVING wins > losses`

Find teams that have won all their home games.

`SELECT home, COUNT(CASE WHEN hscore > vscore THEN 1 END) AS wins,
COUNT(CASE WHEN hscore < vscore THEN 1 END) AS losses
FROM games
GROUP BY home
HAVING wins = COUNT(home) AND losses = 0;`

Table 1.2: Continuing our session with ChatGPT (the model produced responses are in red)

We further type three more queries at the prompt, and the chatbot returns a correct SQL query in each case as shown in Table 1.2. A few observations that can be made here. First, the chatbot remembers the context, and it assumes that this is a continuation of the previous session, hence it uses the tables given earlier to write the query. Second, it is less sensitive to spelling mistakes in the input. For example, the query “Write a SQLite query to teams ...” is missing the term “find”, but the chatbot still gives a correct SQL query. Third, the chatbot is less sensitive to the prompt formatting. Our last query starts with a query in English without the prefix given in the previous queries. The chatbot knows from the context that a SQL query is sought, and it provides that query. Finally, all generated SQL queries will compute correct results, though some generated SQL queries will give additional information that is not asked for. For example, the last two queries only ask for the team names, but the generated SQL queries will give not only the teams names but also the numbers of their wins and losses. Also for the last query, the condition “losses = 0” is redundant.

ChatGPT is not developed in a vacuum. It builds upon many previous models and techniques in the field of natural language processing and code generation including the past work on natural language interface to databases (NLDBs). Some key innovations that have contributed to the development of ChatGPT include the progress in

4 1. OVERVIEW

neural network architectures, such as transformer models, and the use of large amounts of training data to improve language understanding. Furthermore, techniques such as transfer learning and fine-tuning have helped the model to adapt to a wider range of domains and tasks.

1.2 NLIDBS IN THE WILD

With the major progress we are witnessing in NLIDB models especially LLM-based chatbots such as ChatGPT, it is expected that these new models will be incorporated in many commercial products and interfaces to databases. Microsoft has already reported plans to add ChatGPT to its Office 365 products including Word and Powerpoint as well as to its Bing search engine [The New York Times, 2023]. There is no reason for not adding ChatGPT to Microsoft SQL server and other commercial database systems.

Even before the introduction of powerful LLM-based chatbots, increasing the number of commercial offerings have been supporting natural language queries in some form to enable business users to ask questions and query data in natural language. For instance, among the 20 vendor mentioned in the Gartner Magic Quadrant for Analytics and BI Platforms 2022, over half provide NLIDB support in some capacity. In this section, we discuss key aspects of NLIDB support with concrete examples from existing commercial offerings. Our goal is to help the readers to gain a holistic high-level understanding of key design decisions for NLIDBs in the wild and to acquire the necessary knowledge and framework to evaluate and compare such offerings on their own. ¹

Language Support

We can characterize a NLIDB as supporting *controlled natural languages* (CNLs), if it restricts the grammar and vocabulary of its input, and supporting (*arbitrary*) *natural language*, if it does not. The benefit of using CNLs is to syntactically constraint the input to reduce or eliminate ambiguity and to reduce the complexity of understanding a full natural language [Schwitter, 2010]. Figure 1.2 illustrates part of a comprehensive documentation on keywords and formats for a CNL. Many commercial offerings do not explicitly impose such constraints and opt to assist query input into queries that the systems can understand.

We can also categorize a NLIDB as *monolingual* or *multilingual*, depending on whether it supports one natural language only or not. Many NLIDBs in commercial offerings are monolingual (usually English). While few do provide multilingual support: Oracle Analytics, for instance, supports as many as 28 languages [Oracle, 2021].

¹Our goal for this section is not to provide a comprehensive review of individual NLIDB in commercial offerings. We have selected the specific examples used in this section from online documentations of various commercial offerings for illustration purpose. The usage of an example from a particular commercial offering does not imply endorsement of the offering.

Query scope	What is supported	Examples
Chart a measure	Measure across a specified dimension's members Syntax: [Measure] by [Dimension]	Show sales by region
	Measure aggregated by a single dimension member Syntax: [Measure] for [Member]	Show sales for John Smith
	Specifying the dimension of a member Syntax: [Measure] for [Member] in [Dimension]	Show sales for John Smith in sales representatives
	Specifying multiple measures Syntax: [Measure] and /, [Measure] by [Dimension]	Show sales and discounts by region Show sales, discounts by region
	Specifying multiple dimensions Syntax: [Measure] by [Dimension] and /, [Dimension]	Show sales by region and priority Show sales by region, priority
	Specifying multiple dimension members Syntax: [Measure] for [Member] and /, [Member]	Show sales for Canada and USA Show sales for Canada, USA
Rank and sort	Show top N values for the specified measure Syntax: top/highest/Largest/best N [Measure] by [Dimension]	Show top five Texas sales by priority
	Show bottom N values for the specified measure Syntax: lowest/least/smallest/worst N [Measure] by [Dimension]	Show lowest/least/smallest/worst five Texas sales by priority

Figure 1.2: Example Controlled Natural Language Support: SAP Search to Insight

Assisting Query Input

NLIDB support in commercial offerings often provide a combination of the following common assistance features to speed up query formulation and to reduce input errors.

Autocomplete Autocomplete (also known as *typeahead*) predicts the rest of the input as a user is typing. It guides the user to provide the keywords and terms that the system already understands.

Autocorrect Autocorrect automatically correct common spelling or typing errors and provide acronym and synonym matching.

Query Suggestion Query suggestions provide list of possible queries that users can select from as they type to reduce query input efforts and help users find queries that the system can understand.

Query Recommendation Query recommendation recommend queries automatically determined or manually specified in advance for a given dataset. It facilitates the data exploration process of users, by guiding those with limited knowledge of the system or the data with ideas of what types of queries they can ask as well as providing more experienced users with promising directions with interesting results or queries that may be hard to phrase.

Expressivity

We can categorize expressivities of NLIDBs along the following dimensions.

Query Complexity Query complexity supported by a NLIDB depends on the type and the specific subset of query language(s) that it can translate questions into. The most popular underlying query languages includes SQL and SPARQL. Some support customized query languages specific to the underlying data store (e.g. Alibaba Cloud TableStore [Alibaba, 2021]). Some may support questions over multiple data stores with different query languages. Almost all systems support simple selection (e.g. ‘show rev-

6 1. OVERVIEW

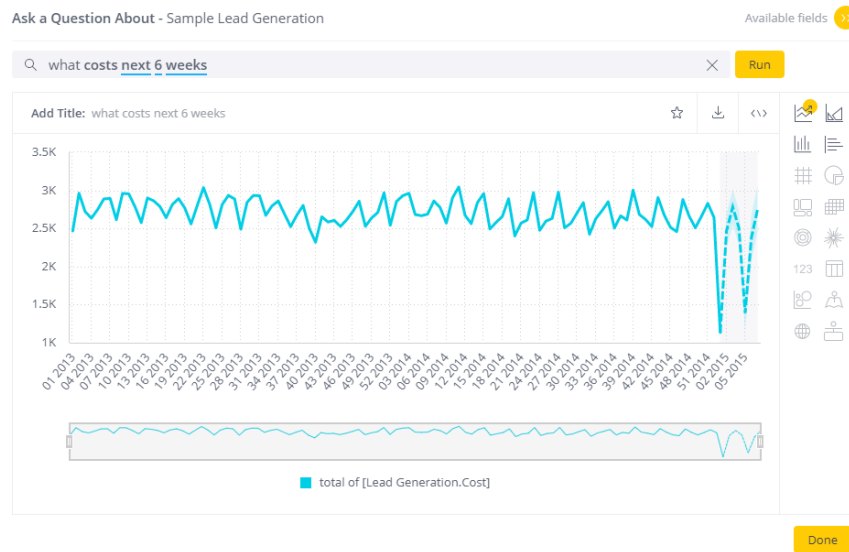


Figure 1.3: Sample Forecast Query: Sisense Simply Ask

enue’), filtering (e.g. ‘in US’), aggregation (e.g. ‘for the past 5 years’), and ranking/sorting (e.g. ‘by amount’). However more complex cases requiring nested query (e.g. “Show me countries sort by total of sales”) are often not supported.

Data Function A NLIDB may allow users to specify other types of data functions besides queries. The data functions can be built-in functions or user defined functions. Figure 1.3 shows an example question that corresponds to the built-in *forecast* function.

Visual Control A NLIDB may allow users to specify which visual to use for result display as part of the question.

Customizability

To ensure NLIDB support works well on specific customer data, NLIDBs need to enable users to customize the support either explicitly via customization or implicitly via learning from interactions. Common features to support customizability include:

Synonym Synonym support is the most common feature to enable customizability. It allows the underlying NLIDB to expand its vocabulary capacity and understand more terms. A NLIDB may allow a user to explicitly specify synonyms, learn from user interactions such as query reformulation, and leverage external knowledge such as WordNet Miller [1995].

Built-in Domain Support A NLIDB may also be customized in advance to understand language in important domains for its customers. For instance, as illustrated by Figure 1.4, the NLIDB understands that “best-selling” means highest revenue in “What



Figure 1.4: Example Query: Amazon QuickSight Q

are the best-selling categories in California?" with customization for sales, marketing, and retail domains.

1.3 WHAT'S AHEAD

This book provides a comprehensive review of the key concepts in natural language interfaces to databases including query understanding and meaning representation, translating queries to structured forms, data models and query languages, evaluation metrics and benchmarks, translating structured data to text and the issues surrounding interactivity. With the relevant topics spread over multiple research communities including database systems, natural language processing, machine learning, human-computer interaction, and visualization, our objective is to provide adequate coverage of these topics, possibly bridging some of the gaps and disconnects that may hinder further developments. The book covers some of the technological progress that have led to more powerful NLIDB interfaces. It also highlights the key underlying concepts and some of the challenges and research opportunities.

The rest of the book is organized into the following three parts.

The first part consists of Chapters 2 and 3 that provides necessary introduction to NLIDBs and key concepts in databases. Specifically, Chapter 2 presents an anatomy of NLIDB and discusses the key components of a conventional pipeline-based approach and some common approaches to build a basic pipeline. Chapter 3 introduces data models and query languages with a focus on the relational model that has been dominant both in industry and the research community. It highlights the key components of SQL, in terms of their functions and the difficulty levels they introduce, as well as some issues in optimizing and evaluating queries.

8 1. OVERVIEW

The second part of the book covers core parts of an NLIDB: text-to-data and data-to-text. Chapter 4 presents the first part—text to data. This component maps a natural language utterance to a formal query language such as SQL. We provide a comprehensive review of the relevant concepts and key component. Evaluating text-to-data models that are expected to work cross-domains and with different database schemes has been a challenge. Chapter 5 reviews some of the evaluation methodologies that have been developed including emerging evaluation metrics and benchmarks. A NLIDB is not limited to mapping natural language utterances to formal languages that can be executed to pull data from a database. It often includes translating or describing the results returned in a tabular form to natural language. Chapter 6 present a comprehensive review on this topic of data-to-text generation.

Finally, Chapter 7 presents interactivity, a key aspect of an NLIDB that affects its effectiveness and usefulness. In this chapter, we introduce different dimensions of interactivity and the related applicable techniques from spell-correction to conversational NLIDBs.