Handbook of Research on Natural Language Processing and Smart Service Systems

Rodolfo Abraham Pazos-Rangel
Tecnológico Nacional de México, Mexico & Instituto Tecnológico de Ciudad Madero, Mexico

Rogelio Florencia-Juarez
Universidad Autónoma de Ciudad Juárez, Mexico

Mario Andrés Paredes-Valverde
Tecnológico Nacional de México, Mexico & Instituto Tecnológico de Orizaba, Mexico

Gilberto Rivera
Universidad Autónoma de Ciudad Juárez, Mexico

IGI Global
PUBLISHER of TIMELY KNOWLEDGE

A volume in the Advances in Computational Intelligence and Robotics (ACIR) Book Series

Table of Contents

Chapter 1
Natural Language Interfaces to Databases: A Survey on Recent Advances

Rodolfo A. Pazos-Rangel
Tecnológico Nacional de México, Mexico & Instituto Tecnológico de Ciudad Madero, Mexico

Gilberto Rivera
https://orcid.org/0000-0002-2365-4651
Universidad Autónoma de Ciudad Juárez, Mexico

José A. Martínez F.
Tecnológico Nacional de México, Mexico & Instituto Tecnológico de Ciudad Madero, Mexico

Juana Gaspar
https://orcid.org/0000-0002-9762-2660
Tecnológico Nacional de México, Mexico & Instituto Tecnológico de Ciudad Madero, Mexico

Rogelio Florencia-Juárez
https://orcid.org/0000-0002-5208-6577
Universidad Autónoma de Ciudad Juárez, Mexico

ABSTRACT

This chapter consists of an update of a previous publication. Specifically, the chapter aims at describing the most decisive advances in NLIDBs of this decade. Unlike many surveys on NLIDBs, for this chapter, the NLIDBs will be selected according to three relevance criteria: performance (i.e., percentage of correctly answered queries), soundness of the experimental evaluation, and the number of citations. To this end, the chapter will also include a brief review of the most widely used performance measures and query corpora for testing NLIDBs.

DOI: 10.4018/978-1-7998-4730-4.ch001

INTRODUCTION

In the last decades, the volume of information has grown exponentially. For manipulating such vast amounts of information, databases have been widely used by businesses and organizations. For accessing database information, different types of software tools have been developed. One type of such tools are database query languages; for example, SQL, which allows users to access data with ample flexibility, because of the high expressiveness of SQL. Unfortunately, SQL is a computer language that is difficult to utilize by users that are not computer professionals.

In order to facilitate casual and inexperienced users accessing database information, graphical form-based applications have been developed. These tools are very easy to use; however, they do not offer flexibility for accessing information in ways different from those for which they are developed.

Natural language interfaces to databases (NLIDBs) are software applications that allow inexperienced users to formulate queries in natural language for obtaining information stored in databases. NLIDBs have the advantages of both types of database querying tools: they are easy to use and offer high flexibility for accessing information.

Several surveys on NLIDBs have been published; some of the most important and recent are the following:

1. Natural language interfaces to databases - An introduction by Androutsopoulos (1995).
2. Natural language interface for database: A brief review by Nihalani (2011).
3. A survey of natural language interface to database management system by Sujatha (2012).
4. Natural language interfaces to databases: An analysis of the state of the art by Pazos (2013).
5. Natural language interface to databases: A survey by Tyagi (2014).

The purpose of this chapter is to describe the most relevant advances in NLIDBs of this decade. Unlike many surveys on NLIDBs, for this chapter, the NLIDBs have been selected according to three relevance criteria: performance (i.e., percentage of correctly answered queries), soundness of the experimental evaluation, and the number of citations. To this end, the chapter will also include a brief review of the most widely used query corpora for testing NLIDBs. The focus of this chapter is on approaches that translate queries in natural language to SQL expressions; so, other database query languages are out of the scope (e.g., Porras, Florencia-Juárez, Rivera & García, 2018).

BACKGROUND

NLIDBs are software applications that allow users to formulate queries in natural language for obtaining information stored in databases. This is accomplished by translating a natural language expression into an SQL statement. Unfortunately, the translation from a natural language query to SQL is an extremely complex problem. This difficulty explains the slow development of NLIDB technology, which is summarized next.

Chomsky (1957) published a monograph titled *Syntactic Structures*, which has been considered a landmark of modern linguistics. He proposed a formal approach to natural language syntax, which consists of symbols and rules and is the origin of the constituency grammar approach. In the decades of the 60s and 70s, the first natural language querying systems were developed, and they were basically interfaces for expert systems implemented for specific domains. Some of the most famous are BASEBALL (Green, Wolf, Chomsky, & Laughery, 1961) and LUNAR (Woods, Kaplan, & Webber, 1972). Most of those NLIDBs were developed for a particular database, and consequently, they could not be easily modified for querying different databases. These systems are called domain-dependent NLIDBs, and many of them achieved good results: accuracy (percentage of correctly translated queries) of around 95%.

The development of an NLIDB for a specific database was time consuming, and it was similar to developing an information system before database management systems were available. Therefore, the next step in NLIDB technology was to develop domain-independent systems, i.e., NLIDBs that can be used for querying many different databases. LADDER (Hendrix, Sacerdoti, Sagalowicz, & Slocum, 1978) is considered one of the first systems that could be configured for querying different databases. It was until 1990 that IBM introduced SAA LanguageAcess (its first commercial NLIDB), which was withdrawn from marketing in 1993. In 2000 Microsoft included English Query in SQL Server 2000, which had a similar fate because it was no longer included in SQL Server 2005. Developing domain-independent systems has proven to be very difficult: they still have deficiencies in the translation process and have attained accuracies in the interval of 80-90%.

NATURAL LANGUAGE INTERFACES TO DATABASES

This section includes brief descriptions of the most relevant NLIDBs developed from 2010 to 2019. Descriptions are grouped according to the approach used for the interfaces: neural networks, syntax based, semantic grammar, and pattern matching, as well as systems for languages different from English. Additionally, the performance obtained (mainly accuracy or precision) and the databases/benchmarks used for testing are mentioned. Additionally, for each approach, the systems are ordered from the most recent to the oldest.

Neural network NLIDBs use deep neural networks (based on recurrent neural networks), which are powerful machine learning techniques that have been used for natural language processing. In particular, a type of deep neural networks, called sequence-to-sequence recurrent neural networks, has been used for translating from a natural language query to SQL. It is important to mention that, like statistical approaches, interfaces based on neural networks have to be trained with very large datasets (tens of thousands of sentences for training the neural network and hundreds of thousands for testing).

Syntax-based systems use a grammar that consists of symbols and rules, which are applied to a natural language expression for determining its structure and grouping syntactically related words. The following is an example of a simple grammar for parsing *Which river passes through Illinois?* (Pazos et al., 2013):

S → *WQ VP*
Wh → “what” | “which”
WQ → *Wh* “river” | *Wh* “state”
VP → *V ValN*
V → “passes through” | “borders”
ValN → “Illinois” | “Missouri” | “Indiana”

Semantic grammar systems are similar to syntax based systems, except that nonterminal symbols may be different from syntactic concepts (noun phrases, prepositional phrases). The following is an example of a semantic grammar (Pazos et al., 2013):

S → *River_question Flow_through*
River_question → “which river”
Flow_through → “passes through” *State*
State → “Illinois” | “Missouri” | “Indiana”

Pattern matching uses a simple technique based on patterns or rules that are applied to the user query. The following is an example of a rule that could be used:

“Which river passes through” <*State*>

The corresponding SQL statement (see Figure 3) is

SELECT *State.state_name*
FROM *State*, *RiverState*, *River*
WHERE *State.abreviation* = *RiverState.abreviation*
AND *RiverState.river_id* = *River.river_id*
AND *State.state_name* = <*State*>

Neural Network Systems

NADAQ System

NADAQ is an NLIDB that combines deep learning and traditional techniques of SQL parsing (Xu et al., 2019). To this end, the system adds to the decoding phase new dimensions of schema-understanding bits and includes new neurons controlled by a finite state automaton for supervising the grammatical states in the decoder part. Additionally, the NLIDB includes a technique that allows the neural network to reject user queries that are irrelevant for the database domain and suggests candidate queries in natural language.

NADAQ consists of three major modules, which are described next:

1. **Data Storage:** This module includes MySQL as a database management system, from which it extracts table metadata for training the translation model, and which processes SQL statements for presenting results to the users.
2. **Model Management:** This module constitutes the NLIDB kernel, which uses several models for bidirectional translation between natural language and SQL, as well as models for rejecting irrelevant user queries. The module provides information from the models to the User Interface module.
3. **User Interface:** This module consists of the interfaces for user-computer interaction.

The most important module of NADAQ is the User Interface, which performs most of the translation processes. The main components are explained next:

1. **Speech Recognition:** The task of this component is converting a spoken user query into text. To this end, it uses the voice-to-text translator iFlytek. Additionally, it provides manual correction so that the user can review the translator output to adapt it to his/her intention.
2. **Translation:** This component uses a machine learning model based on a recurrent neural network coder-decoder, which is a state-of-the-art technique for machine translation. The innovation of this component consists in the integration of hidden states to the model by using a finite state machine for supervising the grammar states for SQL parsing. These hidden states are useful for discarding invalid output words in the decoder part and providing useful suggestions for improving neural-network training.
3. **Rejection:** This component allows NADAQ to reject incoherent user questions. In order to determine the relevance of user queries to the database, the system includes a rejection model in addition to the translation model. Rejection decisions are made based on the uncertainty of the translation model when choosing tables, columns, and search conditions for the SQL statements.
4. **Recommendation:** This component increases the effectiveness of the interface by providing candidate queries to the user for refinement and selection. For helping users without SQL knowledge, the interface translates the candidate queries to natural language, so that users can easily understand the exact meaning (according to the database domain) of the candidate queries and improving the effectiveness of the user-computer interaction.

NADAQ was evaluated using three databases: MAS (Microsoft Academic Search), IMDb, and Geobase. The tests involved the comparison of three methods: convolutional neural network machine, attention-based sequence-to-sequence machine translation, and semantic parsing model with feedback. The NLIDB obtained F1 scores of 83.9% for Geobase and greater than 80% for IMDb.

Cross-Domain NLI Based on Adversarial Text Method

The NLIDB uses a general-purpose question tagging method and a multi-lingual neural network translation model that allows obtaining domain independence (Wang, 2019). For question tagging, each domain is treated equally by using a recurrent neural network. An approach is proposed where different types of natural language queries and different domains share the same components. To this end, the NLIDB performs a preprocessing that consists in separating the domain-specific information from the query.

Given an NL-SQL pair, the key approach consists in inserting predesigned symbols and annotating DB elements (tables, columns, values, keywords, etc.) mentioned in the question for treating each sample (of different domains and types) uniformly. The approach used by the NLIDB is described next:

1. The NLIDB includes a binary classifier BC, which detects elements for predicting if a data element *e* is present in a question *q* corresponding to an SQL statement *p* according to the semantic meaning of the question. The classifier takes *q* and *e*, without referencing *p*.
2. The system looks for the most influential phrase in the query by using gradient-based adversarial text methods.
3. Symbols are inserted in question *q* for annotating phrases that describe DB elements, which are denoted by *q'*.

4. A multi-lingual cross-domain sequence-to-sequence (seq2seq) model is constructed for translating *q'* to *p'*, where *p'* denotes a query where the DB elements are replaced by SQL symbols inserted in *q*.
5. The symbols inserted are replaced by DB elements for generating the original query; i.e., conversion from *p'* to *p*.

This NLIDB has an encoder that uses a stacked bi-directional multi-layer recurrent neural network. It uses a prefix symbol for dealing with different types of queries and treating each type equally.

Experiments were carried out for this NLIDB using the WikiSQL, OVERNIGHT, and Geoquery880 datasets. The metrics used were query-match accuracy denoted by Acc_{qm}, and the execution results Acc_{ex}. For WikiSQL, the interface obtained an Acc_{qm} of 74.5% and an Acc_{ex} of 82.7%, for OVERNIGHT attained an Acc_{qm} of 76.8%, and for Geoquery880 obtained an Acc_{qm} of 84.1%.

DBPal System

DBPal is based on deep learning models for achieving more robust natural language query understanding in two ways (Utama, 2018). First, DBPal uses a deep model for translating NL questions to SQL, making the translation process more robust to wording variations. DBPal provides a learned auto-completion model that suggests partial extensions of queries to users when formulating questions. Second, DBPal has two important features that are based on neural network models: robust query translation and interactive auto-completion.

DBPal consists of two major components, which are explained next.

1. **Neural query translation:** For achieving robust query translation, DBPal proposes a translation method based on a sequence-to-sequence recurrent neural network model. The robustness of the translation process allows to effectively map natural language varying expressions to predefined relational database operations. An important challenge for NLIDBs that use neural networks is to select a comprehensive training set. The main innovation of this NLIDB is a synthetic generation approach, which takes as input the database schema with minimal annotations and generates a large set of natural language questions and their translation to SQL statements. The generation of the training set consists of two steps: generator and augmentation. The first step uses the database schema and a set of base templates that describe NL-SQL pairs and slot-filling dictionaries for generating from 1 to 2 million pairs. The second step automatically increases the initial set of NL-SQL pairs, using existing language models for automatically modifying the NL part of each pair by using different linguistic variations.
2. **Interactive auto-completion:** DBPal provides a real-time auto-completion tool and question suggestion for helping users that are not familiar with the database schema, thus helping them to write complex queries.

For comparing DBPal versus other approaches, the Geoquery benchmark was used, which has been utilized for evaluating other NLIDBs. Additionally, for testing linguistic variations, another benchmark was generated called Patients, which is a database of hospital patients, which consists of one table and 290 queries. Also, a comparison versus NaLIR and NSP was conducted. The accuracies attained by DBPal for the Patients and Geoquery benchmarks were 75.93% and 48.9%, respectively.

Syntax-Based Systems

NaLIR System

NaLIR (Natural Language Interface to Relational database) is an interactive NLIDB that explains the user how the interface interprets his/her question step by step (Li, 2017). When ambiguities are detected, the interface shows the user various interpretations with explanations for the user to choose from, which allows to solve ambiguities by interacting with the user. A training example is collected each time the user makes a choice and confirms its interpretation.

This system consists of three components, which are described next:

1. **Query interpretation:** This component includes a parse tree node mapper and a structure adjustor, which performs the interpretation of the natural language query and the representation of the interpretation as a query tree.
2. **Interactive communicator:** The task of this component is to handle the interaction with the user in order to make sure that the resulting interpretation is correct.
3. **Query tree translator:** This component carries out the translation of a query tree to an SQL statement and sends the SQL query to a database management system.

More specifically, the query interpretation component consists of three modules, which are explained next:

1. **Dependency parser:** The system uses a dependency parser (Stanford Parser) for generating a parse tree from the natural language query. In this tree, each node represents a word/phrase in the query, and each edge is a linguistic dependency relationship between two words/phrases.
2. **Parse tree node mapper:** This module identifies the tree nodes that can be mapped to SQL components and tokenizes them. Additionally, several nodes may have various mappings, which causes ambiguities when interpreting these nodes. For such nodes, the parse tree node mapper sends the best mapping to the parse tree structure adjustor and transmits all candidate mappings to the interactive communicator.
3. **Parse tree structure adjustor:** This module verifies the correctness of the parse tree, specifically, if the tree is coherent with the database schema and there are no ambiguities in the interpretation. In case the parse is incoherent or ambiguous, the system adjusts the tree structure in two steps. In the first step, the tree nodes are reformulated to make the tree coherent with the semantic coverage of the NLIDB. If there exist several correct candidate trees, the best one is selected for the second step. In this step, the selected parse tree is semantically analyzed, and implicit nodes are inserted to make it more semantically coherent. This process is performed under user supervision.

NaLIR was evaluated using 98 queries of the Microsoft Academic Search dataset, and it achieved an accuracy of 89.79% (88 correct answers).

NLI Based on Layered Architecture

This NLIDB is based on an approach that uses a layered architecture (Pazos et al., 2016). The election of this method originates from the following premise: *translation from a natural language query to an SQL statement is an extremely complex problem*. Systems whose design is based on functionality layers (like the OSI model for communication networks) provide the flexibility and modularity for implementing more complex processing strategies than systems designed otherwise. A layered architecture is recommended for dealing with complex problems; therefore, in this NLIDB, each functionality layer deals with a different problem in the translation process.

The Semantic Information Dictionary (SID) is the keystone of the NLIDB since it stores the information necessary for the interface to interpret a query. Initially, the system carries out an automatic configuration, which populates the SID based on the descriptions of DB tables and columns and the information of relations between tables, which are stored in the data dictionary of the DB management system. Additionally, the SID can keep information on words and phrases that refer to tables, columns, relations between tables, imprecise values, alias values, which allows to have the necessary information and facilitates query interpretation.

The core of the interface consists of three layers, whose description is the following:

1. **Lexical Analysis:** The task of this layer is to divide the user query into tokens, search query words in the lexicon and assign (one or more) POS tags to the words found. In case a word is not found in the lexicon, it is considered as a possible search value.
2. **Syntactic Analysis:** This layer consists of a shallow parser that uses a heuristics for determining one syntactic category for those words with multiple categories and ignoring irrelevant words.
3. **Semantic Analysis:** This is the most important layer, and it performs several tasks for understanding the user query and translating it to SQL.

The Semantic Analysis layer is constituted by five sub-layers that are described next:

1. **Treatment of imprecise and alias values:** This sub-layer detects and deals with words that denote imprecise values (i.e., words that represent value ranges, such as afternoon, evening) and aliases (i.e., words for referring to search values, such as noon, couple, fifth, or Philly instead of Philadelphia). To this end, the process scans each word of the input question and searches the SID to determine if the word is declared as an imprecise or alias value.
2. **Identification of tables and columns:** This sub-layer is responsible for identifying the DB tables and columns referred to by words/phrases in the user query, which can be nominal, verbal, adjectival, or prepositional. Specifically, this sub-layer scans each word/phrase of the input question and searches the SID to determine if the word is associated to a table or column.
3. **Identification of the Select and Where phrases:** From the identification of tables, columns, and search values, this sub-layer uses a heuristics for determining the segments of the question that constitute the Select and Where phrases. To this end, each search value is associated to a column according to the proximity and coincidence of data type. The pairs column-search value constitute the WHERE clause of the SQL statement and the remaining columns constitute the SELECT clause.
4. **Treatment of aggregate functions and grouping:** The task of this sub-layer is to identify and deal with the words/phrases of the query used for referring to aggregate functions and grouping

(Group By); for example, words such as average, how many, minimal, maximal, smallest, largest, first, best, for each, etc. This process is carried out by scanning each word of the input question and searching the SID to determine if the word/phrase is associated with an aggregate function or grouping clause. Since the values stored in different columns may be of different natures, different words/phrases are associated to each column.

5. **Determining implicit joins:** For constructing the SQL statement, it is necessary that the graph consisting of the tables involved in the query and the join conditions (search conditions constituted by a column in one table and a column in another table) be a connected graph. When the graph does not satisfy this condition, this sub-layer generates a connected graph by using a heuristics that adds a minimal number of join conditions. Once a connected graph is constructed, the generation of the SQL statement is straightforward.

An experimental evaluation of this NLIDB was conducted using 71 queries for the ATIS database, and it achieved an accuracy of 90% when configured by the implementers (Pazos et al., 2016). Additionally, comparative tests were performed versus ELF and C-Phrase and using the Geoquery250 benchmark; the accuracies obtained were 56.4% for the NLIDB, 35.6% for ELF, and 56.4% for C-Phrase. There are very few publications that report NLIDB performance when the interfaces are not configured by the implementers. An experiment was carried out with two groups of undergraduate students, who configured the interface for the ATIS database, and the accuracies attained were 44.69% for one group and 77.05% for the other. Finally, a Wizard was developed for semi-automatically fine-tuning the configuration. An experiment was performed with another two groups of undergraduate students, who used the Wizard, and the accuracies obtained were 80.53% and 84.82%.

NALI System

NALI is an NLIDB which uses methods for simplifying the configuration without reducing linguistic coverage and accuracy (Mvumbi, 2016). To this end, it uses two authoring frameworks for reducing the work needed for configuring the system for querying different databases.

The first authoring framework is called top-down, and it uses an unannotated corpus of sample natural language queries for extracting lexical terms for simplifying the NLIDB configuration. This strategy reduces the work for configuring the interface by automatically including the semantic information for verbs in negative form, comparative and superlative adjectives in the configuration model.

The second authoring framework is called bottom-up, and it examines the possibility of constructing a configuration model without manual intervention using the information from the database schema and a dictionary.

NALI uses SQL as output language and English as input language. This system assumes that natural language queries are written without spelling and grammatical errors. The syntactic parser is based on a symbolic method for analyzing natural language queries.

The process for translating a question to SQL consists of four main phases, which are described next:

1. **Lexical analysis:** The task of this phase is to scan the question tokens and perform POS tagging, lemmatization, and named entity recognition.
2. **Syntactic analysis:** This phase uses a dependency parser (Stanford Parser) for generating a parse tree from the natural language query.

3. **Semantic analysis:** This phase is responsible for translating the question to an intermediate representation language, which uses first-order logic to express the question meaning.
4. **SQL translation:** This phase constructs the SQL statement from the logical query.

NALI was evaluated using Geoquery250 and attained an accuracy of 74.5% and a precision of 77.4%.

Ontology-Based NLI to Relational DBs

This NLIDB is based on a generic system that consists of several phases and uses an ontology implemented for a customer database (Sujatha, Raju, & Viswanadha, 2016). This system allows accessing information independently of the underlying database. Additionally, the design of the interface allows the scalability and robustness of the system. Word sense disambiguation is performed by using n-grams.

The proposed approach of this NLIDB takes a natural language query and translates it to an SQL statement by using six phases, which are described next.

1. **Stop word removal:** This phase removes stop words from the natural language query according to a predefined list of stop words.
2. **Stemming:** The task of this phase is to determine the root words of the remaining words.
3. **Content word extraction:** This phase assigns POS tags to words by using a natural language toolkit.
4. **Syntactic analysis:** This phase is responsible for parsing the question using a top-down parser. Parsing is performed by applying syntactic rules expressed in Backus-Naur Form.
5. **Semantic analysis:** This phase uses an ontology and n-grams. The ambiguity of a word meaning is resolved by using n-grams and the ontology constructed from the database schema.
6. **Candidate query formulation:** This phase uses the EFFECN algorithm, which performs the division of the natural language question, joining of tables, and selection of multiple tables and columns according to the search conditions specified in the question.

This NLIDB was evaluated using a set of 100 queries to a customer database, and it obtained an accuracy of 84% and a precision of 86%.

Query Builder Based on Dependency Parsing

The main objective of this NLIDB is to allow users to access information stored in a database, without the need of learning a database query language (Kokare & Wanjale, 2015). Constituency and dependency parsing are two techniques widely used in natural language processing. The NLIDB uses dependency parsing for extracting POS tags and typed dependencies. In dependency parsing, the parse tree connects words according to the relation among words. Each node in the tree represents a word, and the children of a node are words that depend on the parent. The labels of the arcs describe the relationship between parent and child.

The translation of a natural language query to SQL is described next:

1. **Lexical analysis:** The task of this phase is to scan the user question for detecting stop words and punctuation marks, which are discarded. Next, the question is separated into tokens.

2. **Syntactic analysis:** This phase uses a dependency parser (Stanford Parser) for generating a parse tree from the user query. Nouns, adjectives, etc. are related pairwise for constituting the arcs of the dependency tree. Additionally, POS tags and typed dependencies are determined.
3. **Semantic analysis:** This phase is responsible for analyzing the typed dependencies for determining the meaning of the question; specifically, the tokens and nouns are mapped to database tables, columns, and search values. Next, a logic query is generated by including the tables, attributes, and search values.
4. **Translation:** This phase translates the natural language query to SQL using the tables, columns, and values previously determined. Finally, the SQL statement is sent to the database management system for returning the results to the user.

Additionally, the NLIDB uses a buffering strategy that stores user questions and the corresponding SQL translations, so that when a previously processed question is detected, the NLIDB uses the stored SQL statement for avoiding all the translation process.

The accuracy reported for this NLIDB is 91.66%. However, the benchmark used for the evaluation is not specified.

Restricted NL Querying of Clinical DBs

The NLIDB uses an approach based on the Top-k algorithm for translating queries in restricted natural language to SQL (Safari & Patrick, 2014). This interface was designed for querying a specific purpose database of a Clinical Information System (CIS) by using a special-purpose language (CliniDAL) for clinical data analytics, which has six classes of query templates. The mapping and translation algorithms are generic, and therefore, they can be used for querying clinical databases designed in any of the three data models: Entity-Relationship (ER), Entity-Attribute-Value (EAV) and XML.

This NLIDB allows a user to compose a question using the CliniDAL restricted natural language, without requiring any knowledge of the CIS database schema, SQL or XML. CliniDAL is a generic query language, and its associated processes for parsing, mapping, translation, and interpretation of temporal expressions are generic and do not depend on the CIS.

The main components of the NLIDB are explained next:

1. **Query Processor:** This component takes a query expressed in CliniDAL as input and processes it by using its sub-components (Parser, Categorizer and Optimizer) for generating a parse tree of the query. Afterward, the parse tree is processed by the Query Translator.
2. **Query Translator:** This component translates the parse tree of a CliniDAL query to SQL, and it consists of four sub-components. The first subcomponent is Mapper, which tries to map the tokens detected in the CliniDAL query to CIS database elements (tables and columns) using the similarity-based algorithm Top-k along with some NLP tools that include tokenization, abbreviation expansion and lemmatization for preparing information for the automatic mapping. The Translator sub-component performs two classes of translations related to the general CIS data model. If the CIS uses an EAV or ER data model, the CliniDAL query is translated to SQL, while if the CIS stores XML documents, the query is translated to XML. The Temporal Analyzer finds and maps the temporal entity (database table) corresponding to the mapped terms of the query to the data elements of the CIS model.

The NLIDB was evaluated using a database of a Clinical Information System and a corpus of 108 queries, and it obtained an accuracy above 84%.

AskMe System

AskMe is a domain-independent NLIDB that uses previously proposed approaches, such as an ontology for describing the database schema, a template-based method for the dynamic generation of the lexical analyzer, syntactic parser and semantic analyzer (Llopis & Ferrández, 2013). Additionally, it provides an innovative characteristic: services for generating queries that reduce the learning time for users. The design of AskMe allows it to be automatically reconfigurable for multiple domains while achieving accuracy comparable to domain-specific NLIDBs.

AskMe consists of two main components, which are described next:

1. **Ontology builder:** After connecting to a database, AskMe looks in the ontology repository for an ontology that describes the database schema. The repository consists of a dictionary that contains ontology references for any pair <server, database> for which the interface has been used. If the ontology for the database is not in the repository, then the system automatically extracts information on tables, columns, and relations from the database schema for building the ontology.
2. **Dynamic parser generator:** This component automatically creates the lexical, syntactic parser, and semantic analyzer, which allows to interpret natural language queries and to translate them into SQL statements for being executed by the database management system. This component has three sub-components: Lexicon, Syntactic parser, and Semantic Analyzer. The lexicon consists of the set of words/phrases that are used in questions for referring to database tables and columns. The NLIDB uses a Link Grammar Parser for parsing operations. The semantic parser uses semantic templates that are filled with the concepts defined in the database ontology.

AskMe was evaluated using the ATIS database and a set of the 448 queries in the ATIS "Scoring Set A". The NLIDB achieved an accuracy of 94.8%.

Semantic Grammar Systems

Intelligent System for Relational DBs

The NLIDB has a general architecture for an intelligent database system, including an implementation that provides domain independence (Gunjal, Rathod, & Pise, 2017). Another feature of this system is that it can be easily configured. The interface uses a semantic matching technique for converting a natural language query to SQL using a dictionary and a set of production rules. The dictionary consists of semantic sets for tables and columns. The SQL query generated by the NLIDB is executed, and the result is presented to the user. This interface was tested using the Northwind database and a Suppliers-Parts database.

The NLIDB is domain independent, which is achieved by an automatic configuration process. Additionally, the interface can be easily configured; to this end, it uses a set of metadata and a semantic set for tables and columns. The system has an intelligent layer that allows to query any database. The layer carries out the processing of information and allows to answer a large variety of questions.

The main functionality is based on semantic sets and rules, which can be modified by the database administrator. The proposed system consists of two modules, which are explained next:

1. **Preprocessor:** The task of this module is to generate the domain dictionary, which is built automatically, and it also generates rules, which are used by a semantic parser, The rules are based on the database schema, WordNet and feedback from the database administrator. The administrator can add, modify, and delete rules.
2. **Run time processor:** This module uses the rules and tries to match words of the user question to predefined data structures, tables, and columns of the database schema. The rules describe the relations between the table and its attributes.

This NLIDB was evaluated using the Northwind database and the Suppliers and Parts database, and a group of five students was asked to formulate queries in English for the two databases. The query sets consisted of 40 questions for Northwind and 20 questions for Suppliers and Parts. The results from the evaluation were: accuracy of 70% for Northwind and accuracy of 75% for Suppliers and Parts.

Pattern-Matching Systems

nQuery System

nQuery is a domain-independent NLIDB that is based on an approach that focuses on incorporating complex natural language requests (questions and data manipulation operations) together with simple requests (Sukthankar, Maharnawar, Deshmukh, Haribhakta, & Kamble, 2017). The interface allows to process requests that involve aggregate functions, multiple conditions in the Where clause, and clauses such as Order by, Group by and Having. The system has been developed for the MySQL database management system.

nQuery translates requests to SQL before retrieving data from the database. This system focuses on retrieving data, but it also allows the translation of other data manipulation operations (Insert, Delete, and Update). However, the interface translates requests that can be processed by MySQL in order to reduce the complexity of the database requests.

The NLIDB takes as input a natural language request, which is translated to SQL through several phases, which are explained next:

1. **Tokenize and tag:** This phase divides the request into tokens and assigns them POS tags by using the NLTK tokenizer package.
2. **Analyze tagged tokens:** This phase scans the tagged tokens and generates a noun map and a list of verbs. Additionally, the type of SQL statement (Select, Insert, Delete, and Update) is determined.
3. **Map to table names and attributes:** This phase uses the noun map and verb list for generating the table set, which specifies the tables that will be needed for building the SQL statement. This strategy is based on the observation that table names are usually referred to by nouns and verbs in requests. Furthermore, the noun map is used to determine the table columns that are needed in the SQL expression.

4. **Filter redundancy and finalize clause mapping:** This phase obtains information for the Group by and Having clauses from information from previous phases and the basic rules of SQL. In addition, redundant tables and attributes are removed using some filter algorithms.
5. **SQL formation:** This phase selects the appropriate SQL statement template according to the type of SQL statement determined in the second phase. Finally, the SQL statement is generated by filling in the selected template the information on the clauses previously gathered and tables and columns stored in the table and attribute map.

nQuery was evaluated using a corpus of synthetic requests to a bank database and a university database, which respectively have 11 and 6 tables. The NLIDB performance was assessed with 75 requests for the university DB and 50 requests for the bank DB, and it obtained approximately an accuracy of 86%.

Aneesah System

Aneesah is an NLIDB based on the combination of a pattern matching approach and dialog interaction with the user for dealing with natural language complexities and ambiguities for dynamically generating SQL statements (Shabaz, O'Shea, Crockett, & Latham, 2015). The NLIDB has conversational capabilities for providing an interactive and friendly environment to help users to access information in relational databases.

The NLIDDB architecture was designed using a pattern matching technique. Additionally, Aneesah implements a conversational agent based on a scripting language, a knowledge base, and an SQL query engine. The interface has a modular architecture that provides flexibility for querying databases of different domains by configuring the NLIDB. The architecture consists of three components, which are explained next:

Component 1: This component is constituted by a Conversation Manager, User Interface, Temporary Memory, and Conversational Agent (constituted by Controller, Pattern Matching Engine, Pattern Matching Scripting Language, and Response Analyzer). The Controller plans and conducts the interaction with the user for guiding him/her in specifying the information the user wants from the database. The Pattern Matching Engine determines the coincidence of user questions with the scripts in the system knowledge base. Additionally, the NLIDB uses a Pattern Matching Scripting Language that allows dialog with the user.

Component 2: This component consists of a Knowledge Base, which allows Aneesah to interact with the database to be queried (a sales history database), and it can be configured for interacting with different databases.

Component 3: This component consists of an SQL Query Engine, which is constituted by an SQL Configurator, SQL Execution, and SQL Analyzer. This component retrieves information from the database.

This NLIDB was evaluated using a sales history database. In the experiments, two groups of participants were used: group A consisted of participants without SQL skills, while group B consisted of participants familiar with SQL. The overall accuracy was 85.01%, and the overall precision was 92.96%.

Systems Based on Other Approaches

Transfer-Learnable NLIDB

This work proposes an NLIDB that is domain independent and transferable to other databases, which is achieved by learning one model that can be applied to any other relational database (Wang, Tian, Xiong, Wang, & Ku, 2018). The approach adopts the principle of separating data and database schema and adding support for the particularities and complexity of natural language. Specifically, the strategy consists in separating the idiosyncrasy of natural language and focusing on the semantics of SQL queries in order to develop a domain-independent and transferable NLIDB.

For obtaining this objective, the information is separated into specific components: the database schema and the use of natural language specific to the schema. The approach used consists of three stages: conversion of a natural language question q to its annotated form q_a, use of a sequence-to-sequence (seq2seq) model for translating q_a to an annotated SQL statement s_a, and conversion of the annotated SQL statement s_a to a normal SQL statement s.

The annotation of the natural language query is performed for detecting words/phrases in the question for referring to DB elements (tables, columns, and values). However, sometimes words/phrases for referring to DB elements depend largely on the context, and sometimes they are not explicitly specified (semantic ellipsis).

The NLIDB uses the information of the database metadata: database schema, database statistics of each column, and natural language expressions specific for a database, a column, and column values.

This NLIDB was trained and evaluated using the WikiSQL dataset that contains 87,673 natural language queries and their respective translations to SQL and 26,521 tables. The interface attained an accuracy of 82%. For evaluating the transferability aspect, the NLIDB was tested using the OVERNIGHT dataset after being trained with WikiSQL. For this test, the query accuracy was 60% (a reduction of 22%).

NLI Based on Semantic Representations Using Ontologies

This NLIDB is based on a proposed approach that uses semantic representations to model the knowledge of the NLIDB using the Ontology Web Language (OWL). The semantically modeled knowledge allows the system to deal with discourse (a sequence of related questions) and to be used for querying databases of different domains (González et al., 2015).

The most important component of this system is the Customization Module. The configuration of the NLIDB is performed in two phases, which are described next:

1. **Database Schema Extraction:** Knowledge is automatically generated by a configuration module, which extracts metadata from the database schema and identifies the elements that integrate the structure of a relational database, such as table names, column names, data type of the columns and existing relations between tables. To generate the NLIDB knowledge, the database administrator must only indicate to the configuration module the connection parameters to his/her relational database. Subsequently, the configuration module automatically models the knowledge and stores it in an ontology, which is a .owl file.
2. **NLIDB Customization:** Once these elements have been identified, the Customization Module analyzes the name of each of them to generate the vocabulary of the NLIDB, which is extended

using lemmas and synonyms. All these elements are modeled by the Customization Module using the proposed semantic representations.

In order to improve the performance of the NLIDB, the configuration module allows the database administrator to manage the knowledge generated, i.e., add, delete, or update knowledge. In addition, it allows the administrator to define the use of superlatives. Specifically, the configuration process consists of the following steps:

1. Associate language words to database tables and columns, which may occur in user questions for referring to tables and columns.
2. Define superlative words, as well as an indication of whether they refer to a maximal or minimal value and the columns to which the superlative can be applied.
3. Indicate which database columns store information that could be used by users as search values in natural language queries.

Classes, object properties, and data properties were defined to design the proposed semantic representations. Classes to model the name of tables and the name of columns extracted from the relational database were defined. Object properties to semantically relate each of the tables to their respective columns were defined, as well as the relations among tables (for representing referential integrity). Finally, data properties to store the data type of each of the columns were defined. In this way, semantic modeling based on the relational schema of the database is generated.

In order to model the relations between natural language and the semantic schema, classes and object properties were also defined. A class to identify the vocabulary words was defined, as well as an object property to relate each word to the elements of the semantic schema it refers to. A class to model superlative words was also defined. Additionally, data properties to indicate the columns of the database on which the superlative can be used and to define the aggregate function to be applied (max or min) were also defined. In this way, all the semantic knowledge of the NLIDB is generated.

The ontology generated by the configuration module is used by the NLIDB to interpret user queries expressed in natural language, as well as to generate the corresponding SQL query, with which the information requested by the user is extracted from his/her relational database.

An experimental evaluation of this NLIDB was conducted using the Geoquery250 benchmark, which also included other NLIDBs (ELF, NLP-Reduce, and FREyA). The results obtained indicate that the proposed semantic representations, used to model the knowledge of the NLIDB, allowed the interface to obtain a good performance, specifically, an accuracy of 85.2%.

NLI Concordant with Knowledge Base

The NLIDB uses an approach for translating natural language questions to a formal language query by using a graph-based knowledge base, i.e., an ontology (Han, Park, & Park, 2016). This method considers a subgraph of the knowledge base as a formal query. The method is based on the principle that a natural language expression (answerable question) has a one-to-one mapping to a formal query; therefore, the natural language question is translated to a formal query by comparing the question to NL expressions and finding the most adequate. If the confidence level of this comparison is not high enough, the interface rejects the question and does return an answer.

This NLIDB performs the translation of a natural language question to a formal query in two phases, which are explained next.

1. **Generation of system-interpretable expressions from a knowledge base:** This phase prepares all the NL expressions that will be compared to the natural language question. These NL expressions are generated from an ontology. The ontology was designed in such a way that allows each subgraph of the ontology to be expressed in natural language, and at least one NL expression can be generated for each subgraph. For each expression, a sequence of NL tokens is generated for being compared to natural language questions. The sequence is called normalized expression.
2. **Translation of user questions into formal queries by using the generated expressions:** In this phase of the process, a natural language question is translated to a formal query. To this end, the normalized expression that is equivalent to the question is determined. First, for a natural language question, one or more normalized expressions are generated, because of various possible interpretations of the question. Next, pairs of normalized expressions are generated, where one element of the pair is generated from the knowledge base and the other element is derived from the question. Afterward, the meaning of the question is determined by selecting the adequate pair of normalized expressions. Finally, a formal query is generated by using the selected pair.

The NLIDB was evaluated using the Geoquery880 and the Geoquery250 benchmarks and the performance metrics precision and accuracy. For Geoquery880, the interface attained an accuracy of 83.2% and a precision of 86.6%, and for Geoquery250, it achieved an accuracy of 86.6% and a precision of 90.6%.

Question Translation with Generative Parser

This NLIDB takes advantage of the database schema for generating a set of candidate SQL queries, which are classified by an SVM-ranker based on tree kernels (Giordani & Moschitti, 2012). In the generation phase, the system uses lexical dependencies and the database schema for constructing a set of SELECT, FROM, and WHERE clauses and also joins. Additionally, clauses are combined by applying rules and a heuristic weighting mechanism, which generates a list of sorted candidate SQL statements. This method can be recursively applied for processing complex queries that involve nested SELECT statements. Finally, a reranker is used for reordering the list of pairs of questions and candidate SQL statements, where both members are represented by syntactic trees.

Ambiguity and errors may affect the interpretation of the user query; however, information of the database schema can be used for verifying the correctness of the selected interpretation. The strategy consists in generating all the possible SQL queries by using information from the database schema (for example, primary keys, foreign keys, data types, etc.) for selecting the most likely using a ranking method.

The NLIDB translates natural language queries that involve nested SQL queries and complex natural language questions that have subordinate phrases, conjunctions, and negations. To this end, an algorithm is used that is based on coincidences between lexical dependencies and SQL structure, which allows to generate a viable set of queries.

This interface gets the lexical relations of a question by using the Stanford Dependency Parser, which obtains the set of binary word relations between a governor and a dependent (gov, dep), where gov and dep denote a parent node and child node in the parse tree.

This NLIDB was evaluated using three subsets of Geoquery and attained an accuracy of 87.2%, a precision of 82.8%, and an F-measure of 85%.

Systems for Languages Different from English

Hindi NLIDB

This is a domain-specific NLIDB that uses natural language processing techniques (Nichante, Giripunje, Nikam, Arsod, & Sonwane, 2017). The input for this interface is a natural language request (question or data manipulation operation) in Hindi. This request is translated to English using a semantic matching technique. Next, a semantically equivalent SQL statement is generated from the English request, which is sent to a database management system, and the result is presented to the user in the Hindi language. The interface is a domain-specific NLIDB that facilitates access to data, insertion, updating, and deletion of information of a transport database.

The approach used for translating a natural language request to SQL consists of the following steps:

1. Generation of a transport database, which stores information on transport services.
2. Identification of the type of request (select, insert, delete, update, and aggregate functions).
3. Mapping of tokens of the Hindi question to database elements (tables and columns).
4. Generation of SQL statements by mapping input requests with the assistance of stored values in the database.
5. Execution of the SQL statement and presenting the result in the Hindi language.

This NLIDB has two databases: a Compiler Database and a Transport Database. The system also has four important modules: Tokenizer, Mapper, SQL Query Generator, and database management system. The process performed by these modules is explained next:

1. The Tokenizer divides the Hindi language request into tokens and stores these tokens in an array. These Hindi tokens are stored in the lexicon (system dictionary) with their corresponding English words.
2. The Mapper sequentially compares the extracted tokens with tokens stored in the lexicon (system dictionary), where the mapping is performed. Those tokens that match the corresponding English words are saved together with their type, and all the remaining tokens are discarded as useless.
3. With the table and column names, the SQL statement is generated.
4. The generated SQL instruction is executed in the database management system and the answer is presented to the user.

There are no experimental results reported for this NLIDB; the article only mentions that it was tested with a transport database designed by the NLIDB implementer.

GANLIDB System

GANLIDB is an NLIDB that translates queries in the Arabic language to SQL; it is domain independent and can improve through experience its knowledge base (Bais, Machkour, & Koutti, 2016). This NLIDB allows users to access data stored in a database by answering queries in Arabic. The interface uses natural language processing techniques for translating queries into SQL statements. The most important advantage of this system is that it is independent of language, content, and model of the database.

The operation of this interface uses an approach based on an intermediate representation language, which translates a natural language query to a logical query in XML. Expressing the logical query in XML allows the interface to operate independently of language, content, and model (relational, object-oriented, relational-object, XML) of the database.

The architecture of GANLIDB consists of three modules, which are explained next:

1. **Linguistic component:** This module performs several analyses (morphological, syntactic, and semantic) of the natural language query and generates the logical interpretation of the question in an XML expression.
2. **Database knowledge component:** This module translates the logical query resulting from the first module to an SQL statement. Next, the SQL statement is sent to the database management system for generating the result in tabular form. The strategy of separating the database knowledge component from the linguistic component allows the interface to query different domain databases.
3. **Natural language query definition:** This module helps the interface to reuse previously processed queries for reducing translation time.

The NLIDB was evaluated using a corpus of 1,300 synthetic questions. The interface answered correctly 1,166 questions, which is equivalent to an accuracy of 95.1%.

Vietnamese NLIDB

This NLIDB has two main components: Question Analysis module y Result Computing module (Nguyen, Hoang, & Pham, 2012). The first component determines the type of user query and extracts information from it. The second component identifies the information requested by the user y calculates query statistics. This is a domain-specific interface for a survey database for individuals and businesses that want to know economic information from economic surveys.

The NLIDB has two main components, which are explained next:

1. **Question Analysis Component:** In the application domain of the NLIDB, a typical natural language query consists of three parts: question term indicates the type of question, question type specifies the statistical measure that users are interested in, and question information is the type of information requested and that will be used for determining the corresponding table columns that store the information. The purpose of this component is to determine Question Term, Type of Question and Question Information from a user question. Considering that user questions not only specify the requested information, but they include special words and phrases, the NLIDB uses JAPE (Java Annotation Patterns Engine) rules for detecting question terms and integrates the results into a Vietnamese word segmentation VNTokenizer so that the question terms are correctly identified as words. There exist many types of questions in Vietnamese; some of the types are Yes/No, Calculate, Give Reason (Why) y Comparing; therefore, for treating these types, the system uses a statistical question answering system that satisfies user's need of information stored in the database.

2. **Result Computing Component:** This component determines the database columns that correspond to the question information part of the user query. This system analyzes the survey questionnaires and sorts the information of each question of the survey; this process is called Relevant Data Retrieval. This system has a synonym dictionary for the economic and statistical domain.

This NLIDB was evaluated using a database of economic surveys. The corpus of queries consists of 500 questions formulated by users, 300 of these questions were used for training the system and 200 questions were used for evaluation. Out of the 200 questions, 157 were correctly answered, which is equivalent to an accuracy of 78.5%.

COMMERCIAL NATURAL LANGUAGE INTERFACES TO DATABASES

Available Commercial Natural Language Interfaces

Access ELF

Access ELF is an existing commercial NLIDB for translating natural language queries to SQL for Access databases (ELF, 2009). It is considered one of the best commercial interfaces. It is important to mention that, since ELF can only be used for Access, it cannot be used for large databases. It has several important characteristics:

1. **Domain independence:** Once ELF is installed on a computer, it is able to interact with any database (after configuring the interface).
2. **Automatic configuration:** The system can automatically obtain the structure of the database; therefore, its initial configuration is easy and simple. The automatic configuration examines the words in the names of tables and columns, and it uses them for generating the dictionary. These words are called synonyms and are used by ELF as references for tables and columns when they occur in natural language queries. The automatic configuration also stores for each column its data type and the table it belongs to.
3. **Configuration edition:** The configuration is performed by assigning new synonyms to tables and columns or by assigning synonyms to the words that have already been related to tables and columns.
4. **Database semantics:** During the analysis of the user query, ELF examines the terms (called synonyms) used for describing database tables and columns, and it uses its dictionary for trying to predict the synonyms used in questions.

The evaluation described by Conlon, Conlon and James (2004) involved users who are human resource professionals (administrators and staff). The success rate reported (presumably recall) was 70–80%. Additionally, ELF was evaluated using the Northwind database and achieved 91% of accuracy (Githiari, 2014).

EasyAsk

EasyAsk is an existing commercial NLIDB that is used for querying eCommerce databases (FinancesOnline, 2019). This is a software search tool that integrates natural language technology (Quiri) and analyses. Users can utilize keywords or terms to filter search results. This system provides product concepts, which yield specific groups of products that match the description of user requests. Additionally, this interface provides assistance when formulating queries by offering options to users when writing their questions.

The most important features are semantic processing that relates various text descriptions to concepts, automatic word stemming and spell-correction, automatic association of attributes to product concepts, and relaxation (ignoring irrelevant or unknown words).

A major component of EasyASk is Quiri, which is a natural language technology that combines linguistic processing with the understanding of data. Quiri divides a user question into words, and then it groups them into phrases and normalizes the content for interpreting the question. Additionally, it provides spell correction, stemming, and synonyms.

EasyAsk was evaluated using the Northwind database and attained an accuracy of 31% and a precision of 48.4% (Githiari, 2014).

Prototype Natural Language Interfaces

ATHENA

ATHENA is a prototype NLIDB developed by IBM, whose main feature is the ability to process complex nested SQL queries for business applications (Sen et al., 2019). The system uses domain ontologies that describe the semantic entities and their relationships in a domain. This system does not need user training nor feedback.

For processing a user question that involves several nested queries, this interface uses the following components:

1. **Evidence Annotator:** This component scans all the tokens of the natural language query and gathers evidence that one or more ontology elements (concepts, properties, and relations between concepts) have been referred to in the user question. Tokens that are mapped to some ontology elements are called entities (database tables and columns).
2. **Nested Query Detector:** A reasoning submodule obtains information from a linguistic analyzer and semantic annotators for identifying a possible nested query.
3. **Subquery Formation:** In the case of a nested query, this component uses lexicon-based techniques to divide the user question into two segments: the first segment for the outer query, and the second for the inner query. Specifically, this component determines the correct sets of tokens associated to each query by using a set of rules applicable to the query, which use the annotator outputs and domain elements. The result of this component is two queries expressed in Ontology Query Language (OQL).
4. **Subquery Join Condition:** For a nested query, this component generates the join condition that involves the inner and outer queries for constructing the complete query in OQL. Join generation depends on linguistic analysis and domain reasoning.
5. **Query Translator:** The task of his component is to convert an OQL query into an SQL statement.

ATHENA was tested using a FIBEN dataset that contains realistic business intelligence queries and combines data from two different financial sub-domains: SEC (Securities and Exchange Commission) dataset and TPoX (XML transaction processing benchmark). No performance results (accuracy or precision) are reported for this test.

Discontinued Natural Language Interfaces

English Query

English Query (developed by Microsoft) is commercial software that includes a set of tools, which database administrators can use for setting up an NLIDB (Microsoft, 2010). This interface was no longer included in SQL Server 2005. English Query applications allow users to formulate ad hoc questions to a database using English expressions. This system provides an environment for developing an English Query model. However, since databases are different and users formulate a large variety of questions, defining a model for answering user queries can be a complex process.

The task of the English Query engine is to perform the translation from a natural language query to SQL. The engine uses a domain file that contains the database model. The model contains specific information of the database to be queried, specifically, the database schema, a semantic abstraction layer constructed on top of the schema, and a mapping between them. In the model, database tables and columns are represented by entities, and joins are represented by relations. Entities and relations defined in a model allow English Query to translate a question to SQL.

English Query has a wizard that helps to configure the interface by automatically defining some entities and relations based on the structure of the database. Database administrators can define other entities and relations by using the development environment. The environment allows defining joins and change/define entity properties; for example, associated words, entity type, field, help text.

The evaluation of English Query described by Conlon et al. (2004) involved users who are human resource professionals (administrators and staff). The success rate reported (presumably recall) was 70–80%. Additionally, it was evaluated using the Northwind database and obtained an accuracy of 39% and a precision of 46.1% (Githiari, 2014).

SAA LanguageAccess

In 1990 IBM introduced SAA LanguageAccess (IBM, 1990). This interface uses other tools developed by IBM, such as Application System (AS) and Query Management Facility (QMF) for showing results to users. These results are shown in different ways: a pie chart, a table or a histogram. This NLIDB was withdrawn from marketing in 1993.

This interface is based on the use of grammar and dictionaries for translating a natural language query to an SQL statement. To this end, it consists of three main components, which are described next:

1. **Query Interface:** This component is based on AS or QMF, which allows the user to formulate queries in natural language. Additionally, the system uses these tools for showing results to the user.
2. **Natural Language Engine:** The task of this component is to syntactically parse the user question, interpret its meaning, and ensure accurate answers. This component involves the following ele-

ments/aspects. Lexicons or dictionaries for storing basic and domain-specific vocabulary. Syntax or grammar for taking into account the structure of the user question, which is used for performing a syntactic parsing and verifying the correct grammatical form of a question. Semantics for helping to understand the exact meaning of the natural language query; for example, determining the adequate meaning of a word with several meanings. Pragmatics for solving situation-dependent interpretation of the user question; for example, for identifying the references of pronouns in the question.

3. **Customization Tool:** This tool is used for defining the vocabulary utilized by users when formulating questions. Typically, this tool is used by the database administrator, since he/she knows the database structure and the vocabulary used in the organization by specific users. Specifically, it is used for including in the lexicon specific terms and acronyms likely to be used in user questions for referring to database elements (tables and columns).

EVALUATION BENCHMARKS

ATIS Database

ATIS (Air Travel Information Services) database is a relational database that has information on flights, flight fares, aircraft, airlines, airports, and cities in the USA, and ground services. From 1990 to 1994, SRI International used this database for conducting annual evaluations for research sponsored by DARPA for spoken natural language (SRI International, 2019).

The ATIS database contains information obtained from the Official Airline Guide, which is organized as a relational database. The database consists of 28 tables and a total of 125 columns, and its schema is shown in figure 1. The most used corpus to query ATIS for evaluating NLIDBs is the ATIS0 Pilot, which was designed in 1990 by SRI and consists of 2884 of queries. This corpus is available at https://catalog.ldc.upenn.edu/docs/LDC93S4B/trn_prmp.html.

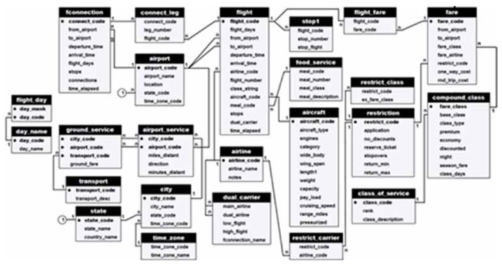
Geobase Database

Geobase is a database that contains geographical information of the United States of America, and it is used by many NLIDBs as a test benchmark since the 90s. It includes information on cities, states, mountains, rivers, lakes, and highways. Originally, Geobase was used by the Geobase system, which was an application example included in the Prolog commercial system, mainly in Turbo Prolog 2.0 version distributed by Borland International (Borland, 1988). It first appeared in 1988 as a deductive database implemented in Prolog, which was complemented with a natural language interface for querying the database.

The schema of the original Geobase is the following:

state(name, abbreviation, capital, area, admit, population, city, city, city, city)
city(state, abbreviation, name, population)
river(name, length, statestringlist)
border(state, abbreviation, statelist)
highlow(state, abbreviation, point, height, point, height)

Figure 1. ATIS database schema

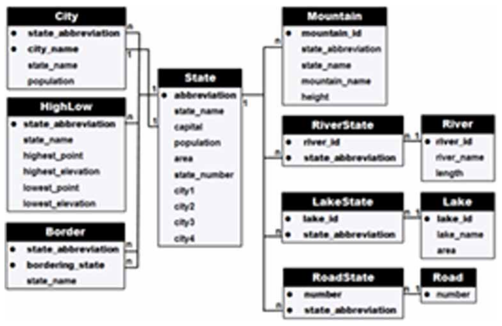


mountain(state, abbreviation, name, height)
lake(name, area, statelist)
road(number, statelist)

Since the 90s, Geobase (sometimes referred to as Geoquery) has been used for evaluating many NLIDBs; some of the most recent are C-Phrase, Precise, and WASP. Therefore, many researchers have adapted Geobase to the relational model.

Unfortunately, there exist different versions of the relational schemas, which differ from the original schema; for example, the addition of an extra column *density* to table *state*, the division of table *highlow* into tables *high* and *low*, the elimination of the four city columns from table *state*. Additionally, most of the versions lack a foreign key from column *capital* to table *city*, which is necessary because every state capital is a city. This situation complicates comparing the performance reported for different NLIDBs, despite being evaluated using Geobase. Figure 2 shows a database schema for Geobase proposed by the authors.

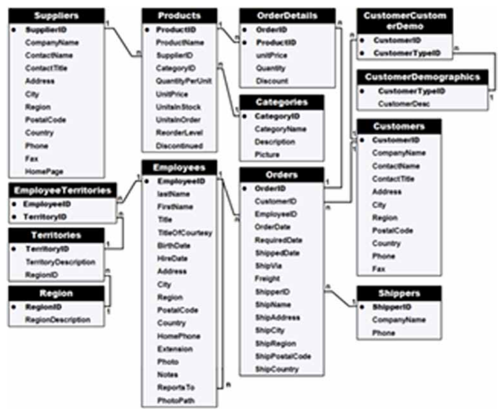
Figure 2. The proposed relational schema for Geobase



There exist two query corpora for testing: Geoquery880 and Geoquery250, which were designed by Mooney for training a system for semantic analysis. These corpora are available at http://www.cs.utexas.edu/users/ml/nldata/geoquery.html.

Northwind Database

Figure 3. Northwind database schema



Northwind is a database used by Microsoft for showing the features of some of its products, including SQL Server and Microsoft Access. The database contains information on sales of the Northwind Traders company, a food import, and export company. This database has 13 tables, as shown in **Figure 3** (Borker, 2006).

Spider Dataset

Spider is a text-to-SQL dataset for large scale semantic analysis and contains complex and cross-domain queries. It was developed at Yale University with the participation of 11 computer science students. It consists of 10,181 natural language questions and 5,693 complex queries on 200 databases with multiple tables that span 138 different domains (Yu et al., 2018).

The query corpus comprises a large variety of clauses and operations: SELECT, WHERE, GROUP BY, HAVING, ORDER BY, LIMIT, JOIN, INTERSECT, EXCEPT, UNION, NOT IN, OR, AND, EXISTS, LIKE, as well as subqueries. Spider is an extensive query corpus designed for training neural networks used in NLIDBs. The corpus comprises 200 databases covering 138 different domains, 10,181 questions, and 5,693 complex SQL queries. This dataset is available at https://github.com/taoyds/spider.

IMDb Database

The Internet Movie Database (IMDb) is an online database that stores information on movies, directors, producers, actors, television shows, and series. IMDb makes available for public access subsets of IMDb data at https://datasets.imdbws.com/. Each dataset is in tsv format and can be accessed using a spreadsheet. Because the information is not compatible with the relational model, some relational database versions have been developed, which vary in the number of tables and columns. One of these is available at https://relational.fit.cvut.cz/dataset/IMDb.

WikiSQL Database

WikiSQL is a dataset published in 2017 by Zhong, Xiong and Socher (2017) for training Seq2SQL, an NLIDB based on deep neural networks. WikiSQL consists of 80,654 annotated natural language queries and SQL queries involving 24,241 tables from Wikipedia. It is important to mention that the WikiSQL database has no foreign keys, and each domain has only one table. This database is available at https://github.com/salesforce/WikiSQL.

SOLUTIONS AND RECOMMENDATIONS

As mentioned before, the slow development of NLIDB technology, from 1961 to 2019, shows that translating a natural language query to SQL has proven to be very difficult. In particular, developing domain-independent systems that achieve accuracies above 95% is an extremely complex problem.

The accuracies reported for neural network NLIDBs are below 85%, which is still far from 95%. The best results for syntax-based interfaces are in the range of 90% to 95% for accuracy. The accuracy obtained by a semantic grammar system is 75%, which is very far from 95%. For pattern-matching interfaces, the accuracies reported are around 85%. The best accuracies obtained by systems based on other approaches is 87%. Finally, the best performance reported for commercial NLIDBs is 80%, which is far from 95%.

The results summarized in the previous paragraph show that more work is needed for increasing the effectiveness of NLIDBs, particularly for neural network systems, which is a new approach for translating from natural language to SQL.

The descriptions of NLIDBs show that there are no widely accepted benchmarks for evaluation, which makes it very difficult to compare the performance of different systems. Although Geobase is one of the most used databases, unfortunately, there exist several versions of the relational schema. Therefore, figure 2 shows a proposed schema for Geobase.

There exist several metrics for measuring NLIDB performance: accuracy (also called recall), precision, and F1. One of the most widely used metrics is precision, which is defined as the percentage of correctly answered queries with respect to the number of translated queries. However, end users are more interested in accuracy, which is the percentage of correctly answered queries with respect to all the input queries. Unfortunately, several systems do not report accuracies, which makes it difficult to compare the performance of NLIDBs that only report precision and/or F1. Therefore, it is recommended that accuracy be always reported.

FUTURE RESEARCH DIRECTIONS

After almost 60 years of development of NLIDB technology, the challenge for achieving accuracies close to 100% still remains. Most of the problems found in translating from natural language are linguistic; for example, modifier attachment, conjunction and disjunction, and semantic ellipsis.

The following example illustrates the problem of modifier attachment: *Which Delta flights depart to Washington at night?* In this sentence, *at night* could be considered as a modifier of *Washington*, which is syntactically correct, but semantically incorrect, because it should be a complement of the verb *depart*. The word *and*, which is a conjunction, is often used to denote logical disjunction, whose meaning is usually difficult to determine. Ellipsis is the omission of important words (in the wording of a natural language query) that are necessary for an NLIDB to understand the query fully. The following query for the ATIS database illustrates this problem: *How many engines does an M80 have?* In this query, a value *M80* is specified; however, no table name is mentioned nor a column name associated to the specified value.

Other problems are related to database schemas and usability. The evaluations reported are for NLIDBs that have been configured by the implementers, who are the experts. However, for practical applications, it is more important the performance obtained when the interface is configured by database administrators. Unfortunately, very few NLIDBs report results of systems configured by users different from the implementers; therefore, more work is needed for addressing this problem.

CONCLUSION

The current challenge for domain-independent NLIDBs consists of developing systems whose accuracy is above 95% and that can be easily configured for obtaining this accuracy.

The development of domain-independent NLIDBs has proven to be more difficult than initially thought. By way of comparison, it has been shown that the problem of determining if a natural language expression is grammatical (i.e., it satisfies the rules of grammar) is an NP-complete problem (Koller & Striegnitz, 2002). Therefore, the syntactic parsing of natural language is an NP-hard problem, whose exact solution requires an algorithm whose computational complexity is exponential.

In order to improve NLIDB performance, the authors propose a layered approach, which is recommended for dealing with complex problems. Systems whose design is based on functionality layers provide the flexibility and modularity for implementing more complex processing strategies than systems designed otherwise. Complex systems that have been developed using a layered architecture are communication networks that use the seven-layered architecture of the OSI model, and database management systems that use the three-level ANSI-SPARC architecture.

The architecture of the NLIDB described by Pazos et al. (2016) has allowed to include more layers for dealing with additional problems. For example, a Wizard for semi-automatically fine-tuning the configuration, which allowed undergraduate students to configure the system for obtaining accuracies in the range of 80.53% to 84.82%.

Another layer has been recently developed for dealing with a difficult semantic ellipsis problem that involves Boolean columns. These are columns that can only store two possible values: true/false, yes/no, 1/0. Natural language queries that involve Boolean columns do not specify search values; for example, consider the following query to the ATIS database *Which flights are in wide body airplanes?* Table *aircraft* has a column *wide_body*, whose values are *YES* or *NO*, but the search value *YES* is not specified in the query.

Finally, a new layer is being developed for querying databases that have design anomalies, such as the absence of primary and foreign keys, use of surrogate keys instead of primary keys, columns for storing aggregate function calculations, repeated columns in two or more tables, tables that are not in second normal form, and tables not in third normal form. Dealing with this problem is important because there are many databases that have design anomalies; therefore, many NLIDBs do not perform correctly with a large number of databases that have this problem.

REFERENCES

Androutsopoulos, I., Ritchie, G. D., & Thanisch, P. (1995). Natural language interfaces to databases – An introduction. *Journal of Natural Language Engineering*, *1*(1), 29–81. doi:10.1017/S135132490000005X

Bais, H., Machkour, M., & Koutti, L. (2016). An independent-domain natural language interface for relational database: Case Arabic language. In *Proceedings of 2016 IEEE/ACS 13th International Conference of Computer Systems and Applications* (pp. 1-7). Agadir. Morocco: IEEE. 10.1109/AICCSA.2016.7945786

Borker, S. (2006). Business intelligence data warehousing an open source approach (Report). Kansas State University, Manhattan, KS.

Borland International. (1988). *Turbo Prolog 2.0 reference guide*. Scotts Valley, Borland International.

Chomsky, N. (1957). Syntactic Structures (463-480). The Hage: Mouton & Co.

Conlon, S. J., Conlon, J. R., & James, T. L. (2004). The economics of natural language interfaces: Natural language processing technology as a scarce resource. *Decision Support Systems*, *38*(1), 141–159. doi:10.1016/S0167-9236(03)00096-4

FinancesOnline. (2019). *EasyAsk review*. Retrieved on December 16, 2019, from https://reviews.financesonline.com/p/easyask/

Giordani, A., & Moschitti, A. (2012). Translating questions to SQL queries with generative parsers discriminatively reranked. In *Proceedings of the International Conference on Computational Linguistics (COLING)* (pp. 401-410). Mumbai, India: Academic Press.

Githiari, L. M. (2014). *Natural Language Access to Relational Databases: An Ontology Concept Mapping (OCM) Approach*. PhD dissertation.

González, J. J., Florencia-Juarez, R., Fraire, H. J., Pazos, R. A., Cruz-Reyes, L., & Gómez, C. (2015). Semantic representations for knowledge modelling of a natural language interface to databases using ontologies. *International Journal of Combinatorial Optimization Problems and Informatics*, *6*(2), 28–42.

Green, B. F., Wolf, A. K., Chomsky, C., & Laughery, K. (1961). BASEBALL: An automatic question answerer. In *Proceedings of the Western Joint Computer Conference* (pp. 207-216). Los Angeles, CA: Academic Press.

Gunjal, U. P., Rathod, V., & Pise, N. N. (2017). An intelligent system for relational databases. *International Journal of Scientific Research (Ahmedabad, India)*, *6*(3), 1546–1550.

Han, Y. J., Park, S. B., & Park, S. Y. (2016). A natural language interface concordant with a knowledge base. *Computational Intelligence and Neuroscience*, *2016*, 1–15. doi:10.1155/2016/9174683 PMID:26904105

Hendrix, G. G., Sacerdoti, E. D., Sagalowicz, D., & Slocum, J. (1978). Natural language interfaces to databases – An introduction. *ACM Transactions on Database Systems*, *3*(2), 105–147. doi:10.1145/320251.320253

IBM. (1990). *IBM SAA LanguageAccess: A straighter way to your information*. Retrieved December 16, 2019, from https://archive.org/details/TNM_IBM_SAA_Language_Access_-_IBM_20171017_0041

International, S. R. I. (2019). *Air Travel Information Service (ATIS)*. Retrieved December 2, 2019, from http://www.ai.sri.com/natural-language/projects/arpa-sls/atis.html

Tyagi, M. (2014). Natural Language Interface to Databases: A Survey. *International Journal of Scientific Research (Ahmedabad, India)*, *3*(5), 1443–1445.

Kokare, R., & Wanjale, K. (2015). A natural language query builder interface for structured databases using dependency parsing. *International Journal of Mathematical Sciences and Computing*, *1*(4), 11–20. doi:10.5815/ijmsc.2015.04.02

Koller, A., & Striegnitz, K. (2002). Generation as dependency parsing. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics* (pp. 17-24). Philadelphia, PA: Academic Press.

Li, F. (2017). *Querying RDBMS using natural language* (Doctoral dissertation). University of Michigan, Ann Arbor, MI.

Llopis, M., & Ferrández, A. (2013). How to make a natural language interface to query databases accessible to everyone: An example. *Computer Standards & Interfaces*, *35*(5), 470–481. doi:10.1016/j.csi.2012.09.005

Microsoft. (2010). *Chapter 32 - English Query Best Practices*. Retrieved December 16, 2019, from http://technet.microsoft.com/es-mx/library/cc917659(en-us,printer).aspx

Mvumbi, T. (2016). *Natural language interface to relational database: A simplified customization approach* (Master thesis). University of Cape Town. Cape Town, South Africa.

Nguyen, D. T., Hoang, T. D., & Pham, S. B. (2012). A Vietnamese natural language interface to database. In *Proceedings of 2012 IEEE Sixth International Conference on Semantic Computing* (pp. 130-133). Palermo, Italy: IEEE. 10.1109/ICSC.2012.33

Nichante, R., Giripunje, S., Nikam, A., Arsod, S., & Sonwane, N. (2017). Hindi language as a graphical user interface to relational database for transport system. *International Research Journal of Engineering and Technology*, *4*(3), 349–353.

Nihalani, N., Silakari, S., & Motwani, M. (2011). Natural language interface for database: A brief review. *International Journal of Computer Science Issues*, *8*(2), 600–608.

Pazos, R. A., Aguirre, M. A., González, J. J., Martínez, J. A., Pérez, J., & Verástegui, A. A. (2016). Comparative study on the customization of natural language interfaces to databases. *SpringePlus*, *5*(553), 1–30. doi:10.118640064-016-2164-y

Pazos, R. A., González, J. J., Aguirre, M. A., Martínez, J. A., & Fraire, H. J. (2013). Natural Language Interfaces to Databases: An Analysis of the State of the Art. In *Recent Advances on Hybrid Intelligent Systems* (pp. 463–480). Springer-Verlag. doi:10.1007/978-3-642-33021-6_36

Porras, J., Florencia-Juárez, R., Rivera, G., & García, V. (2018). Interfaz de lenguaje natural para consultar cubos multidimensionales utilizando procesamiento analítico en línea. *Research in Computing Science*, *147*(6), 153–165. doi:10.13053/rcs-147-6-12

Safari, L., & Patrick, J. D. (2014). Restricted natural language based querying of clinical databases. *Journal of Biomedical Informatics*, *52*, 338–353. doi:10.1016/j.jbi.2014.07.012 PMID:25051402

Sen, J., Ozcan, F., Quamar, A., Stager, G., Mittal, A., Jammi, M., ... Sankaranarayanan, K. (2019). Natural Language Querying of Complex Business Intelligence Queries. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference* (pp. 1997-2000). 10.1145/3299869.3320248

Shabaz, K., O'Shea, J. D., Crockett, K. A., & Latham, A. (2015). Aneesah: A conversational natural language interface to databases. In *Proceedings of the World Congress on Engineering* (pp. 227-232). London, UK: Academic Press.

Software, E. L. F. (2009). *ELF Software Documentation Series*. Retrieved December 16, 2019, from http://www.elfsoft.com/help/accelf/Overview.htm

Sujatha, B., & Raju, S. V. (2016). Ontology based natural language interface for relational databases. *Procedia Computer Science*, *92*, 487–492. doi:10.1016/j.procs.2016.07.372

Sujatha, B., Raju, S. V., & Shaziya, H. (2012). A Survey of Natural Language Interface to Database Management System. *International Journal of Science and Advanced Technology*, *2*(6), 56–60.

Sukthankar, N., Maharnawar, S., Deshmukh, P., Haribhakta, Y., & Kamble, V. (2017). nQuery - A natural language statement to SQL query generator. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, Student Research Workshop* (pp. 17-23). 10.18653/v1/P17-3004

Utama, P., Weir, N., Basık, F., Binnig, C., Cetintemel, U., Hättasch, B., . . . Usta, A. (2018). *DBPal: An end-to-end neural natural language interface for databases*. Retrieved from https://arxiv.org/abs/1804.00401

Wang, W. (2019). A cross-domain natural language interface to databases using adversarial text method. In *Proceedings of the Very Large Data Bases PhD Workshop*. Los Angeles, CA: Academic Press.

Wang, W., Tian, Y., Xiong, H., Wang, H., & Ku, W. (2018). *A transfer-learnable natural language interface for databases*. Retrieved from https://arxiv.org/abs/1809.02649

Woods, W. A., Kaplan, R. M., & Webber, B. N. (1972). The lunar sciences natural language information System (BBN Report 2378). Bolt Beranek and Newman Inc.

Xu, B., Cai, R., Zhang, Z., Yang, X., Hao, Z., Li, Z., & Liang, Z. (2016). NADAQ: Natural language database querying based on deep learning. *IEEE Access: Practical Innovations, Open Solutions*. Advance online publication. doi:10.1109/access.2019.2904720

Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., … Radev, D. R. (2018). *Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task*. Retrieved from https://arxiv.org/abs/1809.08887

Zhong, V., Xiong, C., & Socher, R. (2017). *Seq2SQL: Generating structured queries from natural language using reinforcement learning*. Retrieved from https://arxiv.org/abs/1709.00103