# SpeakQL Natural Language to SQL

*Dhairya* Shah, *Aniruddha* Das, *Aniket* Shahane, *Dharmik* Parikh, and *Pranit* Bari

[1]Dwarkadas J Sanghvi College of Engineering

**Abstract.** Incorporating SQL questions from normal language is a long-standing open issue and has been drawing in extensive intrigue as of late. Natural Language Interface (NLI) is the confluence of Natural Language Processing (NLP) and Human-Computer Interaction, which allows interaction between humans and computers through the utilization of Natural Language. Here we are gonna deal with the problem of automatic generation of Struc- tured Query Language (SQL) queries. SQL is a database language for querying and manipulating relational databases. Despite the spectacular rise in the acceptance of relational databases, there is a fundamental limitaion to the ability to fetch data from those databases. One of the major reasons for this is the fact that the users of these relational databases need to comprehend convoluted structured query languages. In this body of work, we present an interface that allows users to interact with the databases using Natural Lanaguage as opposted to the conventional structure query languages.

## 1 Problem Definition

A vast amount of today's information is stored in relational databases. Writing and executing SQL queries is an integral part of relational database courses. Natural Language Processing (NLP) is one of the most active techniques used in Human- Computer Interaction. It is a branch of Artificial Intelligence (AI) that is used for information retrieval, machine translation and linguistic analysis. The main objective of NLP is to allow communication between humans and computers without memorizing commands and complex procedures. Posing questions to databases in the form of natural language is a very handy and straighforward technique of accessing data, especially for beginners who do not fully comprehend convoluted query languages such as SQL. This system focuses on the solution of the problems aris- ing in the analysis or generation of Natural language text or speech. The main purpose of Natural Language Pro- cessing is that the computer must be able to interpret a speech signal and perform the required action. For casual users who do not understand database query language like SQL, an easy method of data access is asking questions to databases in natural language and we will handle the com- plex queries for them. Hence it is not necessary for the user to have prior knowledge about the SQL queries. NLP is a technique that makes the computer understand the languages used by humans. While natural language may be easy for people to learn and use, it has been proved to be hard for a computer to master. Despite such challenges, natural language processing is regarded as a promising and important endeavour in the field of computer research. The goal is to inculcate the computer with the ability to understand and generate natural language so that the user can address his computer through text as though he was addressing another person. We all want to make better decisions by using data, which means we need to overcome major obstacles such as tedious dashboards or SQL scripts. One way to solve it would be to convert natural language to SQL like Google Translating is translating Japanese to English which is exactly what we aim to achieve.

## 2 Literature Survey

Initially [1] a seq2seq(encoder-decoder) approach along with reinforcement learning to reward the model based on the generated output was used. The input sentence is broken down into tokens which are used to generate the SQL query by the augmented pointer network. The input sequence is the concatenation of the column names, required for the se- lection column and the condition columns of the query, the question, required for the conditions of the query, and the limited vocabulary of the SQL language such as COUNT, SELECT etc. Three functions are utilized in order to produce the SQL query.

Each word is assigned a type as an entity from either the number, column or knowledge graph, by a type aware model [2] . It employs a sketch-based approach and views the task as a slot filling problem. By grouping dif- ferent slots in a reasonable way and capturing relationships between attributes. It uses an input encoder consisting of 2 bi-directional LSTMs to encode word and pair type. To encode word and type pairs of the question, they concatenate embeddings of words and their corresponding types and input them to BI-LSTM.

Natural Language is mapped to executable programs by focusing on Semantic Parsing [3]. This technique employs pointer networks, which creates the SQL query by encoding the question into continous vectors, using three channels. Gradually, the model understands when to generate

an SQL keyword, a cell or a column name.In order to mitigate the ill-formed outcomes, the model integrates column cell relation as well. Pointer networks are originally introduced [4] which takes a sequence of elements as the input and out- puts a sequence of discrete tokens corresponding to posi- tions in the input sequence.

A different approach [6] where instead of generating SQL query it focuses on output cells containing data as answers to the given input natural language question. It leverages semantic parsing for question answering systems over databases. It uses BERT to implement this question answering system. BERT is a new language representation model, which stands for Bidirectional Encoder Representations from Transformers. BERT is a pre-trained deep model which uses a large amount of plain text to pre-train. Subsequently, we can fine-tune the pre-trained BERT representations with an supplementary output layer to produce state-of-the-art models for a wide range of tasks, such as text classification and question answering. First, it introduces the idea of finding the answer from a database without semantic parsing to solve the problem of hard label work for semantic parsing. Sec- ond, it uses a BERT-based model to implement the idea and achieve a baseline level experiment result.

## 3 Proposed Architecture with Modular Description

One of the most common approaches to converting natural language to SQL queries is to use a sequence-to-sequence style model, usually with reinforcement learning. But the most common issue that is faced in this kind of a model is the fact that order of the generated sql query matters. A common question is why does the order matter? - A very easy explanation can be given to explain the issue. Let's take a set of random numbers and we desire the indices of the numbers in such a way that they are sorted according to some given rule. The rule could be outputting them in ascending order, descending order or any arbitrarily fixed permutation. This yields exactly n! permutations, all of which are valid. If we generate our training set by randomly picking out any of those permutations, all the n! final output configurations will have equal probability for the same input vector X. Thus, this formation is much less statistically efficient. Thus it was found that restricting the order as much as possible for the outputs was always a better approach

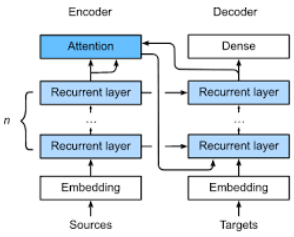


**Figure 1.** Sequence to Sequence model

To mitigate the issues of order in sequence-to-sequence-style models, usually reinforcement learning is used. The model trained using a policy gradient algorithm. But improvement due to the implementation of reinforcement learning is often limited. A sketch-based approach is used in this case to generate an SQL query from a sketch. The most challenging part is to generate the WHERE clause. The issue with sequence-to-sequence decoder is that the prediction of the next token depends on all previously generated tokens. The sketch-based approach uses two devices: sequence-to-set and column attention. The former is designed to predict an unordered set of constraints instead of an ordered one, and the later is designed to capture the dependency relationship defined in the sketch when predicting. SQLNet only needs to fill in the slots in the sketch rather than to predict both the output grammar and the content. The sketch will highly align with the SQL grammar. The prediction of the value of one slot is only conditioned on the values of those slots that it depends on. Each dependency is a directed edge in the sketch. Thus our model can be viewed as a graphical model based on this dependency graph and the query synthesis problem as an interface problem on the graph. To synthesize more complex SQL queries, we simply employ a sketch that supports a richer syntax.

Sequence-to-set: Instead of generating a sequence of column names, we can simply predict which column names appear in this subset of interest. We compute the probability, Pwherecol(col|Q). Two LSTMs encode the column names and the questions do not share their weights. The dimension of column vectors and embeddings of the column name and the natural language question have the same dimension of their hidden states of the LSTM. Column attention: Basically selecting which column is the most relevant to predicting the real column. Eg. "number" is more relevant to predicting the column "no." but the token "player" is more relevant to predicting the column "player".

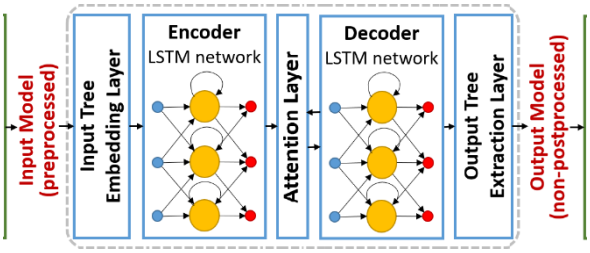


**Figure 2.** Encoder-Decoder with Column Attention Architecture

Selection of column slot: After the computations of the probabilities of columns being included in the set are done, SQLNet needs to decide which columns to include in the WHERE. In particular, a network is used to predict the total number K of columns to be included in the subset, and choose the top-K columns with the highest probability to form the column names in the WHERE clause. Selection of operator slot: For each column in the WHERE clause, predicting the value of its OP slot is a 3-way classification: the model needs to choose from =, <, >. Column attention is also used in selecting the operator for the slot. Selection of value slot: Here we need to use a sequence-to-sequence structure to generate the sub-string. The order of the tokens in the VALUE slot indeed matters. Selection of column slot: The selection of the column slot for the SELECT clause is the same as the WHERE clause except there's only one column.

The column names and natural language descriptions are treated as a sequence of tokens. Each token is represented as a one-hot vector and fed into a word embedding vector before feeding them into the bi-directional LSTM. The GloVe embedding is used in this case. Input encoding model details: Column names and natural language descriptions are treated as a sequence of tokens using the Stanford CoreNLP tokenizer to parse the sentence. Each token - one-hot vector and fed into a word embedding vector before being fed into the bi-directional LSTM. The "GloVe" word embedding is used. Weight sharing details: Components in SQLNet only share the word embedding. The model, also, has multiple LSTMs for predicting different slots in the sketch. Different LSTMs can be used for predicting different slots simultaneously, which will yield a better performance. Training the word embeddings during training yielded better results.

## 4 Expected Outcome

The output expected is the data obtained from the database for the given input. The output will be in the form of table similar to output generated from a SQL query. SQL query generated for the natural language will also be provided. Moreover, user's also have a quick access to recent queries entered by the user. This could as well be a step forward to making database usability accessible to people who have no knowledge how a database query works.

It can handle up to two levels of complexity (nested loops). Outcomes involving algebraic SQL functions like *avg, sum,* etc and most queries work smoothly via this approach. A primary issue with the approach is identifying correct tables to join in cases of semantic ambiguity. In addition to that, the use of well-defined constraints, prevents the system from generating queries involving select all operations using the asterisk (*) operator. While the results returned may still be the same, the query generated would be more elaborate relatively.

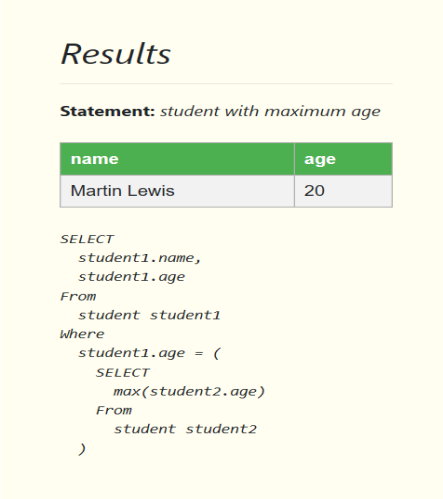


**Figure 3.** Generated Output (1)

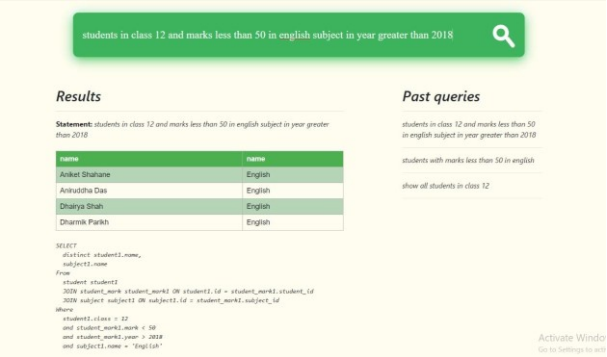


**Figure 4.** Generated Output (2)

## References

[1]Victor Zhong, Caiming Xiong, Richard Socher, Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning (2017)

[2]Wonseok Hwang, Jinyeong Yim, Seunghyun Park, and Minjoon Seo: Exploration on WikiSQL with Table-Aware Word Contextualization. (2019)

[3]Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, Dragomir Radev, TypeSQL: Knowledge-based Type-Aware Neural Text-to-SQL Generation (2018)

[4]Yibo Sun, Duyu Tang, Nan Duan, Jianshu Ji, Guihong Cao, Xiaocheng Feng, Bing Qin, Ting Liu, Ming Zhou Semantic Parsing with Syntax- and Table-Aware SQL Generation (2018)

[5]Yibo Sun, Duyu Tang, Nan Duan, Jianshu Ji, Guihong Cao, Xiaocheng Feng, Bing Qin, Ting Liu, Ming Zhou

Semantic Parsing with Syntax- and Table-Aware SQL Generation (2018)

[6]Po-Sen Huang, Chenglong Wang, Rishabh Singh, Wen-tau Yih, Xiaodong He Natural Language to Struc- tured Query Generation via Meta-Learning (2018)

[7]Tong Guo, Huilin Gao Table2answer: Read the database and answer without SQL (2019)

[8]Ion Androutsopoulos, Graeme D Ritchie, and Peter Thanisch Natural language interfaces to databases–an introduction. (1995)

[9]Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang Semantic parsing on freebase from question-answer pairs (2013)

[10]Ronghang Hu, Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Kate Saenko. Learning to reason: End-to-end module networks for visual question an- swering (2017)

[11]Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In Association for Computational Linguistics (ACL) System Demonstrations (2014)