Using SQL/MED to Query Heterogeneous Data Sources with Alexa Voice Commands

Johannes Schildgen[1] Florian Heinz[1] Andreas Olijnyk[1] Arvid Lindenau[1]

Abstract: Typical Alexa skills and other add-ons for voice assistants need to be custom developed for their one specific use case. This paper presents an approach to map arbitrary data sources (databases, APIs, services) to the relational model by using SQL/MED and to transform voice-based queries into SQL. The key challenges for such a universal skill are to correctly map the natural-language question into a SQL query on the correct source table in the federated database and to convert the result set back to a compact and well-understandable answer.

Keywords: Voice Assistants; SQL/MED; Natural-Language Processing; User interfaces for big data

1 Introduction and Motivation

Speech recognition and voice-based queries have been research topics for many years. While transforming speech into written text has basically reached a good-enough level [Hu17], the big challenge in building voice assistants is understanding what the user wants to know or to do. As there is no universal machine that understands all kinds of voice queries, companies develop individual voice applications for each specific scenario: finding train connections, news, stocks, weather, and so on. Alexa, Siri, and Google Assistant only support a limited set of built-in commands. Nevertheless, they often use a fallback method by doing a traditional web search with the full query and speaking out the text of the most relevant search result. As this approach is very error-prone and does not support customized queries, the typical approach is developing custom add-ons for voice assistants. For Alexa, these add-ons are called skills.

Each skill has to be individually developed by defining a list of example sentences, so-called intents. An intent contains zero or more slots. A slot is like a variable; it has a name and a data type. The skill developers must implement a when-this-then-that behavior of what to answer or do when which intent is called. For example, "Which are the meals in the canteen on <date>". Voice frameworks work in a flexible way so that this intent also matches the following input sentence: "Tell me today's meals in the canteen." In this case, the slot `<date>` will be set to the current date.

Besides these end-user-focused use cases, voice interfaces are increasingly adopted in professional contexts. A recent trend is the use of natural-language interfaces to work with

[1] OTH Regensburg, Postfach 120327, 93025 Regensburg, Germany
{johannes.schildgen,florian.heinz}@oth-regensburg.de; {andreas.olijnyk,arvid.lindenau}@st.oth-regensburg.de

doi:10.18420/BTW2023-53

data-analytics applications [Go20]. These so-called conversational analytics allow users without data-science skills to explore and analyze data. Big business-intelligence platforms like Tableau Ask Data [Ma18] or SAP Conversational AI have been launched in recent years to enable companies with business data to use natural-language queries.

This paper presents an approach that uses the database language SQL as an intermediate layer between Alexa skills and arbitrary data sources. Of course, this approach is not limited to Amazon Alexa; it also works for arbitrary other voice assistants. Our command from above would be translated into the following query: `SELECT name FROM meals WHERE meal_date = current_date;` The table names and column names (`meals`, `name`, `meal_date`) are determined by exploring the database metadata. If the desired data is stored in the tables of a relational database, then this query can simply be sent to that database. If not, our approach uses external tables as proposed by the SQL/MED standard (Management of External Data) [Me01]. Depending on the federated database management system, this concept is also called foreign-table wrappers (PostgreSQL), datalinks (DB2), external tables (Oracle), connect tables (MariaDB) or virtual schemas (Exasol). The idea of SQL/MED is to virtually integrate external data sources, like APIs or NoSQL databases, and let them appear in the database catalog like native tables. Each query on these tables is forwarded to the external system on demand (see Figure 1). In the example above, `meals` could be an external table to the API of OpenMensa.org or a view that joins physical and external tables.

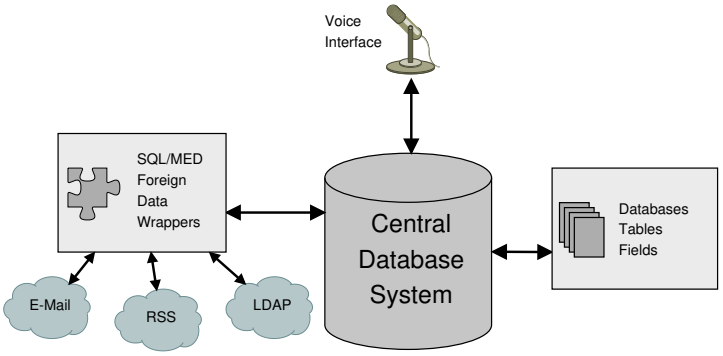

Fig. 1: Database System as the central data source for natural language queries

In this paper, we make the following contributions:

1. We present an SQL/MED-based approach for voice queries on arbitrary data sources.
2. We identify how to transform natural-language queries into valid SQL commands.
3. We transform query results back into a compact and well-understandable voice answer.

The following section shows a brief history of natural-language queries and other related work. In Section 3, we explain our approach for a universal Alexa skill. In Section 4, we present exemplary use cases and a short evaluation. We conclude in Section 5.

2 Related Work

Translating natural language into SQL queries (NL2SQL) has been a research topic for several decades [ART95, LJ14a, LJ14b]. Kim et al. [Ki20] provide an overview of those NL2SQL techniques. They use different benchmarks and automatic and manual matching approaches to evaluate whether natural-language commands are translated into a semantically correct SQL query or not. Some of the systems with the highest accuracy are NSP [Iy17], TypeSQL [Yu18], GNN [BGB19], and IRNet [Gu19]. They all use deep-learning methods to learn cross-domain as well as domain-specific terminologies and relationships. The approach that we used for our prototype implementation uses simple natural-language processing techniques. But it is easy to replace these techniques with others to increase the accuracy.

EchoQuery [Ly16] is a system that focuses on voice inputs and voice outputs to query relational databases. It can be used in an interactive way. The user can refine previous queries with follow-up questions, and the system can ask clarifying questions back to the user. The findings from this approach can easily be applied to our approach as well to make it more interactive. Cyrus [GJ19] is an iPhone app that converts voice commands into SQL queries for teaching purposes. The app can be used to learn SQL. Cyrus is very similar to EchoQuery and also to our approach. But our approach is voice-only; Cyrus displays the query and the result table on the smartphone display. They do not convert the result set into a well-understandable spoken answer.

There is only a little research on one-size-fits-all conversational systems. Haase et al. [Ha17] convert voice commands into SPARQL queries and send them to the Wikidata knowledge graph. However, most skills for Alexa and others are custom-developed for just one single application scenario. Atefi et al. [At20] studied the user reviews of more than 2800 of those custom Alexa skills. They found out that the most frequent complaint is that the content provided by the skill is undesired or uninteresting. The main reason for this is that in custom skills, users often need to ask questions in a prespecified and fixed form. With our approach, we try to solve this problem by building a universal Alexa skill. There, fuzzy matching methods on database metadata allow for a broad diversity of supported queries.

Given that databases are often accessed via an API, some work has been done on querying APIs with natural language (NLI2API). This is often done by training machine-learning models on examples of natural-language commands and their mapping to API calls. In [Su17], the authors proposed a framework to collect high-quality training data and evaluated it using real-world APIs, yielding good results. In follow-up work, an NLI2API system with fine-grained control was tested in a user study [Su18]. The participants had to complete several tasks using text queries, e.g., retrieving emails from their inboxes. The novel approach in this work was to split the queries into so-called modules that correspond to different API parameters. After performing a query, users could remove or add modules to correct their initial query. Eventually, the approach reduced the number of required interactions and task completion time and thus improved the overall user experience.

3 Implementation based on Amazon Alexa

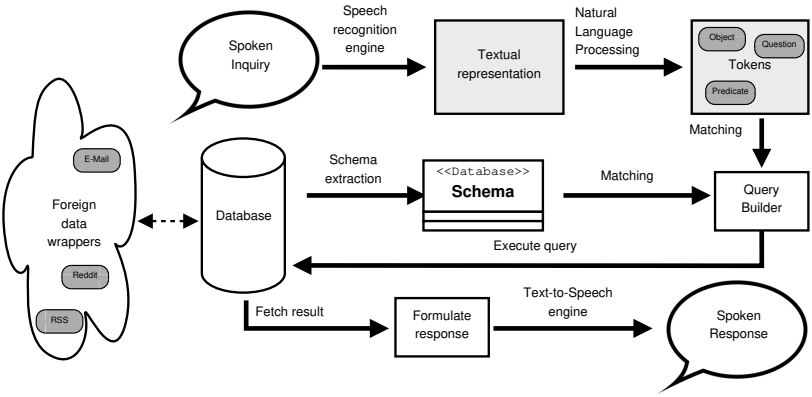

Fig. 2: Process Overview

Figure 2 sketches the coarse overview of the process. The human interface is a speech-recognition service as is provided by, for example, Alexa, Siri, Google Assistant, Cortana, Bixbi, or similar engines. Our reference implementation is basically independent of the speech-recognition service, but we chose Amazon Alexa for our prototype. The Alexa skill transcribes the spoken query into a text string, and this string is fully sent to our Python-based server via HTTPS. The only task of the voice service is the speech-to-text and text-to-speech part.

The main part of the process happens within a server application. When this server starts up, it first connects to the database to discover the database schema. In this step, not only the table and column names are fetched, but also a set of aliases for each of these identifiers is generated. They contain different word variations such as singular/plural forms or synonyms. For this task, string-normalization techniques, thesauruses, dictionaries, and the Python libraries "nltk", "wordnet" and "inflect_engine" are used.

After receiving the inquiry string, also this query is cleaned and normalized. For example, Alexa passes numbers up to ten as words, not as a sequence of digits, which is fixed up in that step, together with several other minor corrections like resolving abbreviations ("who's" → "who is") or question words ("how much" → "price"). These words are then used to try to find the correct parts of the database schema in the following order: (1) direct matches (original identifiers), (2) fuzzy search (using doublemetaphone), (3) matching of aliases, (4) association matching. After that, the matching concepts of the found tokens in the database query have to be determined: table name, column name (projection or selection), column value, and aggregation function. This is performed by heuristics that take into account the position inside the sentence, preceding or following words, and several more criteria. Finally, the query is executed and a natural sounding answer is generated from the result set. So, for example, the following conversations could be happening:

```
> What is the firstname of the employee with employeeid five?
The firstname of the employees with the employeeid 5 is 'Steve'.
```

It is quite useful that the answer always contains parts of the question. If the answer would just be “Steve”, the user cannot be that sure that the question was understood correctly by the system.

Limitations Our prototype supports single-table queries with multiple projection columns in the `SELECT` clause, simple aggregation functions (`COUNT(*)`, `SUM|AVG|MIN|MAX(col)`) without `GROUP BY`, and a `WHERE` clause with one or many (`AND|OR`) predicates. Each predicate consists of a column identifier, a comparison operator (`<`, `<=`, `=`, `!=`, `>`, `>=`, `LIKE`) and a numeric or string literal. These are the same limitations as in the WikiSQL [ZXS17] benchmark. For more complex queries with joins or groupings, the workaround is to create a view and query that view with a voice command.

4 Use Cases

In the previous section, we presented a simple implementation of how a natural-language query can be translated into a SQL query. This allows for voice queries on arbitrary tables in a relational database. This section describes some interesting use cases and evaluates the overall usability of this system. We will not only use native SQL tables for this but we will also use external tables. The idea is to virtually integrate external data so that it can be accessed with SQL. For our prototype, we use PostgreSQL’s foreign data wrappers. One can develop their own wrapper or use one of the available foreign data wrappers for JDBC, NoSQL (MongoDB, Neo4J, Redis, ...), files (CSV, JSON, XML, ...), services (Google Spreadsheet, Open Weather Map, Open Street Map, Twitter, ...) and many more[4].

One useful foreign data wrapper is the *FileGW* module, which spans a bridge between the database and a local file system. After creating the foreign table, a typical command could be: “What is the content of the file with the name memo?”.

Another interesting foreign-data wrapper is the RSS gateway[5]:

```
CREATE FOREIGN TABLE newsfeed (title varchar(800), description varchar(800),
pubDate timestamp, link varchar(800)) server rss_srv options (url '...xml');
```

A command like “What are the titles of the newsfeed of the last two hours?” or “What is the description of the newsfeed with the title ’clean energy’?” can now be used to extract information from arbitrary RSS feed sources.

Many types of information can be represented by a relational schema and the foreign-data-wrapper method is an easy and practical way to not only retrieve that information from a database but also link it with other relations in the database. It is for example possible to create a view that joins several (native or foreign) tables and retrieve information from this view with a spoken inquiry.

[4] `https://wiki.postgresql.org/wiki/Foreign_data_wrappers`
[5] `https://multicorn.org/foreign-data-wrappers/`

External tables in PostgreSQL are not read-only but also writable. This makes it possible to extend our voice assistant by supporting inserts, updates, and deletes. For example, the foreign data wrapper for Philips Hue[6] shows a table with all lightbulbs and their states. A voice command can create an `UPDATE` query to turn on or off one or multiple lights.

Companies are increasingly adopting NLP technologies to power data-driven use cases. Decision makers use KPIs to monitor the current status of manufacturing processes, make analyses like trend predictions or get notified in case of critical deviations. We used our voice-based approach in a proof-of-concept implementation at an automotive company and it showed that it can improve the usability of a data-analytics application in a manufacturing domain. As an alternative to using the web application, users can use voice commands to query an underlying API and retrieve the desired information like "What is the current production status in plant Berlin?". There has been a major challenge in implementing these use cases. Professional environments often have a domain-specific vocabulary that is difficult to integrate into NLP systems. Thus, it is important that the relevant terms are present in the database metadata. Database views can be used to structure the data in a useful way and to name the attributes accordingly, e.g., `production_status`. Once the data is structured in a useful way, the system can be used immediately.

Tests with users showed that the system improves the usability of the underlying KPI system because the information can be easily retrieved in any scenario directly on their phone without the need to have access to a PC.

5 Conclusion

This work shows a general-purpose method to retrieve specific information from an SQL database system. The focus where this work excels is to cope with database systems by only retrieving the database schema; other information is not needed. Nevertheless, the user can intuitively formulate spoken inquiries and the system often does a good job of translating the sentence into an SQL query. For this, fuzzy matching with Doublemetaphone and thesaurus lists is used to find out where the sought information might be stored. External information can be provided to the database by foreign-data-wrapper modules, where a rich amount of existing modules is available, but also developing customized wrappers is not a big task.

Further development should include the automatic joining of tables to retrieve information that cannot be derived from a single table. In the current state of the implementation, one or more views would have to be prepared in advance to achieve such a task. Beyond that, it would be desirable to support some more typical operations and aggregations on both sides - projections and predicates - of a query. As modern AI language models are well-suited for generating SQL queries, one approach could be integrating systems like OpenAI GPT to support more complex queries.

Our prototype's source code is open-source, available at `github.com/fwheinz/datalexa`

[6] `https://github.com/rotten/hue-multicorn-postgresql-fdw`

Bibliography

[ART95] Androutsopoulos, Ion; Ritchie, Graeme D; Thanisch, Peter: Natural language interfaces to databases–an introduction. Natural language engineering, 1(1):29–81, 1995.

[At20] Atefi, Soodeh; Truelove, Andrew; Rheinschmitt, Matheus; Almeida, Eduardo; Ahmed, Iftekhar; Alipour, Amin: Examining user reviews of conversational systems: a case study of Alexa skills. arXiv preprint arXiv:2003.00919, 2020.

[BGB19] Bogin, Ben; Gardner, Matt; Berant, Jonathan: Representing schema structure with graph neural networks for text-to-SQL parsing. arXiv preprint arXiv:1905.06241, 2019.

[GJ19] Godinez, Josue Espinosa; Jamil, Hasan M: Meet Cyrus: the query by voice mobile assistant for the tutoring and formative assessment of SQL learners. In: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing. pp. 2461–2468, 2019.

[Go20] Goasduff, L: , Megatrends Dominate the Gartner Hype Cycle for Artificial Intelligence, 2020.

[Gu19] Guo, Jiaqi; Zhan, Zecheng; Gao, Yan; Xiao, Yan; Lou, Jian-Guang; Liu, Ting; Zhang, Dongmei: Towards complex text-to-sql in cross-domain database with intermediate representation. arXiv preprint arXiv:1905.08205, 2019.

[Ha17] Haase, Peter; Nikolov, Andriy; Trame, Johannes; Kozlov, Artem; Herzig, Daniel M: Alexa, Ask Wikidata! Voice Interaction with Knowledge Graphs using Amazon Alexa. In: ISWC (Posters, Demos & Industry Tracks). 2017.

[Hu17] Huang, Xuedong: Microsoft researchers achieve new conversational speech recognition milestone. Microsoft, August, 2017.

[Iy17] Iyer, Srinivasan; Konstas, Ioannis; Cheung, Alvin; Krishnamurthy, Jayant; Zettlemoyer, Luke: Learning a neural semantic parser from user feedback. arXiv preprint arXiv:1704.08760, 2017.

[Ki20] Kim, Hyeonji; So, Byeong-Hoon; Han, Wook-Shin; Lee, Hongrae: Natural language to SQL: where are we today? Proceedings of the VLDB Endowment, 13(10):1737–1750, 2020.

[LJ14a] Li, Fei; Jagadish, Hosagrahar V: Constructing an interactive natural language interface for relational databases. Proceedings of the VLDB Endowment, 8(1):73–84, 2014.

[LJ14b] Li, Fei; Jagadish, Hosagrahar V: NaLIR: an interactive natural language interface for querying relational databases. In: Proceedings of the 2014 ACM SIGMOD international conference on Management of data. pp. 709–712, 2014.

[Ly16] Lyons, Gabriel; Tran, Vinh; Binnig, Carsten; Cetintemel, Ugur; Kraska, Tim: Making the case for Query-by-Voice with EchoQuery. In: Proceedings of the 2016 International Conference on Management of Data. pp. 2129–2132, 2016.

[Ma18] Markas, Ruhaab: , Ask Data: Simplifying analytics with natural language, 2018.

[Me01] Melton, Jim; Michels, Jan-Eike; Josifovski, Vanja; Kulkarni, Krishna; Schwarz, Peter; Zeidenstein, Kathy: SQL and management of external data. ACM SIGMOD Record, 30(1):70–77, 2001.

[Su17] Su, Yu; Awadallah, Ahmed Hassan; Khabsa, Madian; Pantel, Patrick; Gamon, Michael; Encarnacion, Mark: Building natural language interfaces to web apis. In: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management. pp. 177–186, 2017.

[Su18] Su, Yu; Hassan Awadallah, Ahmed; Wang, Miaosen; White, Ryen W: Natural language interfaces with fine-grained user interaction: A case study on web apis. In: The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval. pp. 855–864, 2018.

[Yu18] Yu, Tao; Li, Zifan; Zhang, Zilin; Zhang, Rui; Radev, Dragomir: Typesql: Knowledge-based type-aware neural text-to-sql generation. arXiv preprint arXiv:1804.09769, 2018.

[ZXS17] Zhong, Victor; Xiong, Caiming; Socher, Richard: Seq2sql: Generating structured queries from natural language using reinforcement learning. arXiv preprint arXiv:1709.00103, 2017.