UNIVERSITY OF CALIFORNIA SAN DIEGO

This is the Title of My Dissertation

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy/Doctor of Musical Arts/
Doctor of Education

in

My Degree Title

by

My Full Legal Name

Committee in charge:

Professor Eta Theta, Chair
Professor Gamma Delta, Co-Chair
Professor Lambda Kappa
Professor Iota Mu
Professor Epsilon Zeta

2018

Copyright

My Full Legal Name, 2018

All rights reserved.

The Dissertation of My Full Legal Name is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2018

DEDICATION

In recognition of reading this manual before beginning to format the doctoral dissertation or master's thesis; for following the instructions written herein; for consulting with OGS Academic Affairs Advisers; and for not relying on other completed manuscripts, this manual is dedicated to all graduate students about to complete the doctoral dissertation or master's thesis.

In recognition that this is my one chance to use whichever justification, spacing, writing style, text size, and/or textfont that I want to while still keeping my headings and margins consistent.

EPIGRAPH

True ease in writing comes from art, not chance,
As those move easiest who have learn'd to dance.
'T is not enough to no harshness gives offence,—
The sound must seem an echo to the sense.

Alexander Pope

You write with ease to show your breeding,
But easy writing's curst hard reading.

Richard Brinsley Sheridan

Writing, at its best, is a lonely life. Organizations for writers palliate the writer's loneliness, but I doubt if they improve his writing. He grows in public stature as he sheds his loneliness and often his work deteriorates. For he does his work alone and if he is a good enough writer he must face eternity, or the lack of it, each day.

Ernest Hemingway

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

PREFACE

Almost nothing is said in the manual about the preface. There is no indication about how it is to be typeset. Given that, one is forced to simply typeset it and hope it is accepted. It is, however, optional and may be omitted.

ACKNOWLEDGEMENTS

I would like to acknowledge Professor Eta Theta for his support as the chair of my committee. Through multiple drafts and many long nights, his guidance has proved to be invaluable.

I would also like to acknowledge the "Smith Clan" of lab 28, without whom my research would have no doubt taken fives times as long. It is their support that helped me in an immeasureable way.

Chapter 2, in full, is a reprint of the material as it appears in Numerical Grid Generational in Computational Fluid Mechanics 2009. Smith, Laura; Smith, Jane D., Pineridge Press, 2009. The dissertation author was the primary investigator and author of this paper.

Chapter 3, in part, has been submitted for publication of the material as it may appear in Education Mechanics, 2009, Smith, Laura; Smith, Jane D., Trailor Press, 2009. The dissertation author was the primary investigator and author of this paper.

Chapter 5, in part is currently being prepared for submission for publication of the material. Smith, Laura; Smith, Jane D. The dissertation author was the primary investigator and author of this material.

VITA

2005	B.S. in Business Management, Biola University, La Mirada, California
2017	M.S. in Management - Manpower Systems Analysis, Naval Postgraduate School, Monterey California
2018	B.S. in Computer Science, California State University - Monterey Bay, Seaside California
2025	Ph.D. in Computer Science, University of California San Diego

PUBLICATIONS

Kyle Luoma and Arun Kumar. "SKALPEL: Schema Knowledge Adjustments for LLM-based NL-to-SQL Performance Enhancements in Large Databases," Under Submission.

Kyle Luoma and Arun Kumar. "SNAILS: Schema Naming Assessments for Improved LLM-Based SQL Inference," Proceedings of the 2025 International Conference on Management of Data (SIGMOD'25), 1875–1900, 2025.

ABSTRACT OF THE DISSERTATION

This is the Title of My Dissertation

by

My Full Legal Name

Doctor of Philosophy/Doctor of Musical Arts/
Doctor of Education in My Degree Title

University of California San Diego, 2018

Professor Eta Theta, Chair
Professor Gamma Delta, Co-Chair

The Abstract begins here. The abstract is limited to 350 words for a doctoral dissertation. It should consist of a short statement of the problem, a brief explanation of the methods and procedures employed in generating the data, and a condensed summary of the findings of the study. The abstract may continute onto a second page if necessary. The text of the abstract must be double spaced.

Introduction

This optional section is barely described in the OGS manual other than saying it is optional and that it appears in the table of contents between the Abstract and the first chapter.

No formatting guidelines appear so presumably, it should be formatted like an ordinary chapter. It should appear after the `\mainmatter` macro because it should start on page 1.

Chapter 1

Introduction

Chapter 2

SpeakQL 2.0 - Implementing and Evaluating a Speech Friendly SQL Dialect

2.1 Introduction

Speech assisted computer interaction has increased in popularity over the past decade as speech recognition capability has improved and ubiquitous mobile computing devices such as tables, smartphones, in-home assistants, and vehicle entertainment systems integrate speech recognition into their core features. The SpeakQL project seeks to explore the possibility of moving the application of speech-based system interfaces from simple consumer-facing applications toward more technical areas such as data extraction from relational data stores.

SpeakQL 2.0 is an extension of previous research that implements a speech + touch interface to enable database querying using mobile devices with touchscreen interfaces. User feedback from usability studies performed on this system have motivated additional research, including the creation of a prototype dialect of SQL that, while retaining the expressive power and formality of SQL, creates a query dictation process that adheres to more natural speech patterns. This paper presents the motivation, justification, ongoing implementation, and ongoing evaluation of the SpeakQL dialect.

2.2 Background

2.2.1 Motivation

Proliferation of Speech-Based Assistance Systems

Speech-based assistance systems have become increasingly prevalent due to the growth of the mobile computing, internet of things, and smart device industries. With speech-based systems becoming more prevalent and accepted, we see opportunities to explore the possibility of applications that target a more technical user base. One such potential application is human-database interaction through spoken query dictation. This application has the potential to reduce barriers to employing data-driven decision making in organizations that often conduct critical operations in remote and/or austere conditions, or whos decision-making workers and analysts operate for long periods of time away from their computer workstations.

Target Users

The SpeakQL voice + touch query system was designed with input from the target user population: data analysts who require ad-hoc query capability while performing tasks in environments where they do not have immediate access to traditional computing equipment, and must rely on a mobile device. In prior SpeakQL work, conversations with practitioners revealed a need for SQL-knowleable analysts to access their data in ad-hoc ways while working in environments where standard keyboard-typed querying is not feasible. [13]. In this paper, the authors identified on-the-go database administrators and data analysts, nurse informaticists, and individuals with motor disabilities as users who could benefit from a speech-based querying system.

As we extend the work of SpeakQL, we also consider practitioners in other industries and domains where data-driven analysis and decision making is currently hindered by data access. This could include organizations with analysts who perform work in austere and remote environments and have unpredictable data needs that require ad-hoc query capabilities such

as military cyber-system operators peforming work at the edge of their tactical networks, and analysts working in field-based industries such as resource extraction, construction, and farming. Providing analysts in these fields with mobile-based ad hoc query capabilities has the potential to improve their ability to increase domain knowledge through hands-on engagement and immersion and increase responsiveness to emergent and unpredictable data needs that arise from field-based scenarios.

Increase Career Accessibility by Advancing Speech-Based Programming

Effective speech-based programming technology also has the potential to increase access to programming careers for individuals with disabilities that impact their ability to type. SQL adheres to a relatively standard structure and use a limited set of keywords as compared to most procedural languages, and it serves as a compelling 'first step' towards a general speech-based programming technology. We believe that continue work on SpeakQL has the potential to open doors to future programmers who would otherwise not have had the opportunity to pursue a programming-based career.

2.2.2 The Structured Query Language (SQL)

SEQUEL and SEQUEL 2

[5] [3] The Structured Query Language (SQL), originally named the Structured English Query language (SEQUEL), was designed to reduce the complexity of retrieving data from relational databases. Originally intended to be used by business professionals and other laypersons who required data access, it was one of the first languages designed for a non-programmer user population [5, 4]. It was also one of the first examples of a language built with user-centric design considerations in mind. Several human factors evaluations were conducted during the early design phase of SQL that lead to future modifications to the language [11].

Human Factors Evaluation of SEQUEL

Human factors evaluations were conducted as part of the SEQUEL language development effort. Usability experiments comprised of teaching SEQUEL to programmer and non-programmer college students. Students were then evaluated on their ability to translate English statements into equivalent queries in both the SEQUEL and SQUARE query languages. The study yielded significant results and associated recommendations including the recommendation to make SEQUEL a layered system consisting of three layers representing increasing levels of complexity, and the recommendation to replace complicated correlation and computed variable syntax with the join feature which most SQL users are familiar with now.

Reisner also discoverd that sources of minor errors when converting English statements into SEQUEL queries included ending errors, spelling errors and synonym errors. These discoveries resulted in the recommendation to incorporate spelling correction, introduce a synonym dictionary to the language syntax, and a create stem-matching procedure as user aids which would enable users to use keywords with various forms of conjugation. [11]

Query complexity has a directly proportional affect on the likelyhood of error occurence during query formulation. The complexity of a query and associated likelihood of error is measured by creating an index of query complexity that is derived from the number of English-to-SEQUEL transformations required to generate the query. These transformations are categorized as either augmentation (a word must be added to generate the query), deletion (a word must be removed from the English statement) and replacement (a word in the English statement must be replaced). [12] Reisner, et al, found that the index of transformational complexity was useful for predicting the likelihood of errors encountered while writing SQL queries.

Grammar and Variants

Numerous variants of SQL syntax exist from multiple vendors and open source projects. While each varient tends to contain implementation-specific features targeted at specific database management systems, most (if not all) generally adhere to the ISO/IEC 9075-1:2016 information

technology standard for SQL [6]. Seven SQL grammars are available under various open source licenses on the ANTLR parser Github repository including: hive, mysql, plsql, postgresql, sqlite, trino, and tsql [1].

All SQL grammars include syntax rules for both *data definition language* (DDL) and *data manipulation language* (DML) statements. DDL statements are intended to enable specification of data structure; and DML statements enable data access and update functions [2]. Data analysts, informaticists, and other data consumers generally make use of DML statements to fulfill data-related needs. DDL statements are generally expressed by database administrators and software developers responsible for designing, implementing, and maintaining data models within database management systems.

2.2.3 SpeakQL 1.0

SpeakQL Implementation

SpeakQL is a speech plus touch query system that enables users to submit spoken queries to an arbitrary database schema using a subset of the standard SQL dialect. The SpeakQL system also makes use of other user input and interaction tools including a novel SQL keyboard that enables users to make corrections or additions to aspoken query by selecting SQL keywords or values using a touch screen. SpeakQL development was motivated by a need for data analysts and other database users to have on-the-go ad hoc query capability that provides unambiguous results. SpeakQL provides this capability in the form of a web-based application that interfaces with cloud-based automatic speech recognition (ASR) services for speech transcription and transforms ASR transcripts into valid SQL queries using a structural and literal determination subroutine. SpeakQL users can modify or append generated queries using a novel touchscreen-based SQL keyboard that contains common DML keywords used to generate select expressions [13, 14].

SpeakQL Usability

A comparative user study where participants with prior SQL knowledge used a tablet device to both type and dictate queries revealed that query dictation using the SpeakQL system

leads to lower times and fewer units of work required to form correct queries [13, 14].

2.2.4 Other Rule-Based Speech-to-SQL Systems

EchoQuery

EchoQuery is a hands-free, stateful, voice-based query system that promotes dialogue as a means for users to express their query intent and resolve errors and ambiguities. Intended to interact with users using an audio-only interface (e.g. Amazon Alexa), EchoQuery introduces a modified SQL syntax that promotes natural-feeling dialogue between the user and the system and allows for the modification of a query state using keywords such as *add* and *drop*, to alter the output of the initial query. The syntax is limited in terms of expressive power, and only provides single-relation query capability.[8]

2.2.5 Natural Language to SQL

There has been a significant increase in spoken query projects that employ a deep-learning natural language-based approach to generating SQL queries[7]. NL-to-SQL systems eliminate the need for query writers to adhere to a strict query language syntax. The burden of formally communicating with the database is transferred from the user to the NL-to-SQL translation system. Employing naturally-spoken English as a means to query relational databases introduces a problem of query ambiguity. In NL-to-SQL systems, ambiguity is introduced through the inherrently ambiguous structure of the English language; and some form of disambiguation or query intent confirmation is required to guarantee a query's result matches the user's intent.

2.3 SpeakQL 2.0: a New Dialect

In this section we introduce the SpeakQL dialect–a dialect intended to make the process of query dictation more natural and less prone to errors caused during the dictation process.

2.3.1 Desiderata

The primary objective of the SpeakQL dialect is to increase ease of dictation while retaining the unambiguity and expressive power of SQL.

Formal Query Language

Natural language ambiguity can be a significant source of query result ambiguity, we seek to overcome this by defining the SpeakQL dialect using a context free grammar. Employing a grammar-based approach places the requirement of forming unambiguous query expressions on the system user. This approach differs from natural language-to-SQL systems that employ various methods to interpret a user's query intent from an unstructured natural language sentence. We feel that this tradeoff between structure and ambiguity is appropriate for our target user population of SQL-experienced analysts; as they will already be experienced with the process of forming structured queries using SQL.

Expressive Power

We wish to retain, but not necessarily increase, the expressive power of SQL. We intend to fulfill this objective primarily by retaining all SQL syntax as legal SpeakQL syntax in the SpeakQL grammar. SpeakQL features are then added as extensions to the existing SQL grammar. This means that if a user cannot form a query due to the limited expressiveness of a particular SpeakQL feature, they may opt instead to dictate the query using standard SQL syntax. Therefore, the objective in relation to expressive power is that all SQL queries are also SpeakQL queries, and that all SpeakQL queries can be unambiguously translated to a corresponding SQL query.

Ease of Dictation

The SpeakQL dialect should improve a user's experience when dictating queries both in measurable terms such as error counts and number of attempts needed, and also in less quantifiable terms such as general user satisfaction, frustration, and perceptions of usefulness.

We believe it is acceptable for the dialect to be more verbose, and as a consequence, result in longer dictation times, so long as the increased verbosity contributes positively toward improving other human factors measures related to query dictation such as reduced error rates and decreased planning or 'think' time required before beginning query dictation.

2.3.2 Cognitive Factors

Human working memory suffers from time decay-based forgetting. The ability to retain information and subsequently recall the information from working memory while simultaneously completing a complex task decreases as task time increases. [15]

Various theories relating to working memory and mental models suggest that problems that require the construction and retention of multiple models within working memory are more difficult than problems that deal with the construction of only one model. [16]

The problems of memory decay and multi-model processing difficulty during task execution are motivation for exploring different SQL query structures that reduce the need to hold keywords and multiple schema information in working memory throughout the duration of the query formulation process.

2.4 SpeakQL Dialect Features

The current variant of the SpeakQL dialect has the following features:

- Keyword Synonyms and Optional Syntax
- Query Structure and Order Variance
- Natural Functions
- Complex Query Unbundling

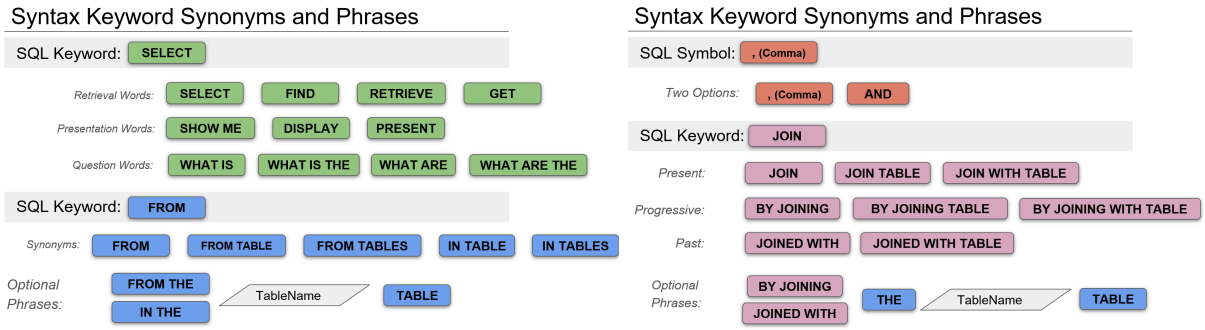


Figure 2.1. SpeakQL Synonyms

2.4.1 Keyword Synonyms and Syntactic Sugar

Synonyms

Keyword synonyms are keywords with meaning equivalent to standard SQL keywords such as select, from, and join. The intent of this feature is to relax the rigidity of the formal SQL syntax and enable users to choose from several options when formulating queries. The standard SQL keywords remain within the vocabulary so users who are highly experienced with SQL may still use SQL syntax. The underlying idea that motivates this feature is that making equivalent keyword synonyms available using formal grammar rules offers more natural speech patterns, particularly in cases when other features such as query reordering or unbundling of queries with multiple relations are used. For example, if a user elects to define the table relation prior to column projection, they may say ”in table x” and follow this statement with the keyword ”find a, b and c” resulting in the statement ”in table x find a, b and c.” This statement is equivalent to the SQL query select a, b, c from table x. Table 2.1 contains a list of SpeakQL keyword synonyms and their associated SQL keywords.

Optional Syntax

SpeakQL allows for the use of optional ’the’ and ’table’ keywords when dictating a table expression. This permits expressions such as ’select * from *the* courseoffering *table*’, where the keywords *the* and *table* are optional additions that exist for the purpose of making table expression statements feel more natural for the speaker.

SQL Keyword	SpeakQL Synonyms
SELECT	Select, Find, Retrieve, Get, Show Me, Display, Present, What Is, What Is The, What Are, What Are The
FROM	From, From table, From Tables, In Table, In Tables
' , ' (Comma)	' , ' (Comma), And
JOIN	Join, Join Table, Join With Table, By Joining, By Joining Table, By Joining With Table, Joined With, Joined With Table

Table 2.1. Keyword Synonyms

Keyword Synonym and Optional Syntax Examples

SpeakQL

*"**Show me** area and wheelchairspaces **in the** room **table** where floor = 2"*

```
SQL: Select area, wheelchairspaces
     from room where floor = 2;
```

SpeakQL:

*"**What is the** count(id) **in the** course **table**"*

```
SQL: Select count(id) from course;
```

2.4.2 Query Structure and Order Variance

A standard SQL select expression must conform to the order *Select -¿ From -¿ Join -¿ Where -¿ Group By -¿ Having -¿ Order By -¿ Limit*. SpeakQL's order variance permits additional orderings (see figure 2.2) of expressions as well as the trailing select modifier clauses (see figure 2.10).

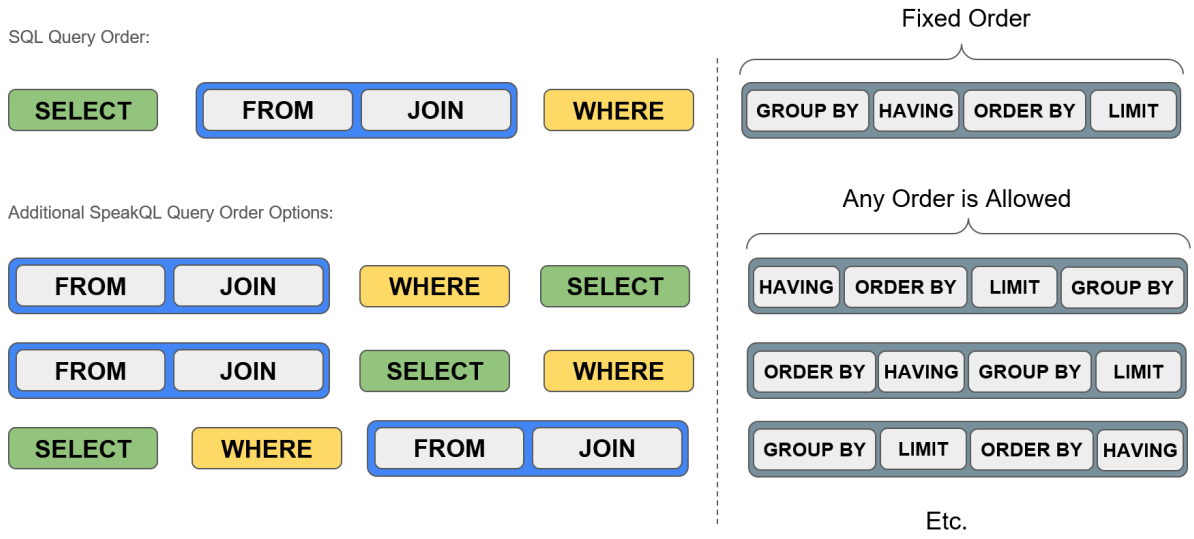


Figure 2.2. Alternate Ordering

As with other features, the alternate ordering feature is optional. Query speakers may choose to conform to the standard SQL order specification. The alternate ordering provides options to query speakers who may be more comfortable with specifying source tables or where predicates prior to column selections or function specifications. We hypothesize that relaxing the order of the modifier clauses *group by, having, order by, and limit* makes the query dictation process more error tolerant by transferring the responsibility for recalling the specific order of the clauses from the query speaker to the SpeakQL translator.S

Alternate Ordering Examples

SpeakQL

*"**From** term **show me distinct** termperiod **table** where year = 2022"*

```
SQL: Select distinct termperiod from term
     where year = 2022;
```

SpeakQL:

*"**In the** courseoffering **table where** capacity ¿ 20 **find** facultyname **and** ondays **limit** 10 **order by** facultyname"*

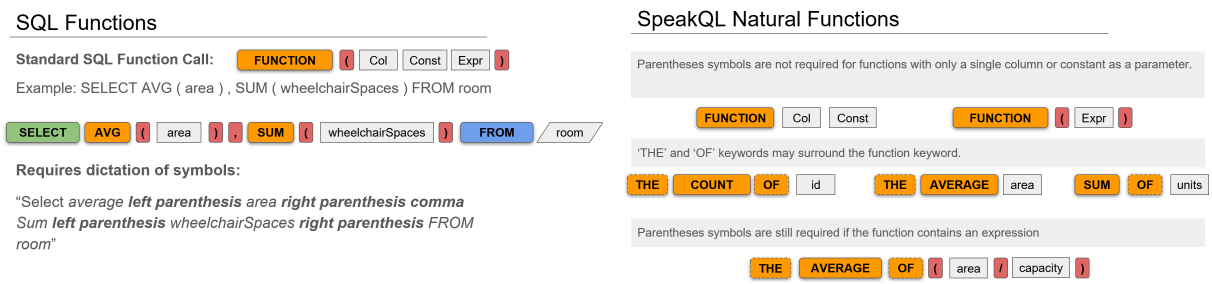


Figure 2.3. Natural Functions

```
SQL: Select facultyname, ondays
     from courseoffering
     where capacity > 20
     order by facultyname limit 10;
```

2.4.3 Natural Functions

The SpeakQL 1.0 grammar includes aggregator functions such as *sum, avg,* and *count*, and query speakers are required to verbalize the parentheses symbols. While parentheses are essential for disambiguation in many traditional cases, for SpeakQL 2.0 we identified a set of function references where parentheses could be omitted without affecting the meaning of the query. Specifically, the SpeakQL dialect permits the expression of functions naturally, that is without verbalizing parenthesis, for functions that have only a constant or single column as an argument (See figure 2.3). The natural function feature also permits optional syntax keywords *the* and *of* that surround the function name.

In cases where a speaker's query intent requires the inclusion of an expression as a function argument, the verbalization of parenthesis remains a requirement. This allows the SpeakQL dialect to retain SQL's capability to pass mathematical, comparative, and subquery expressions as function arguments without these expressions introducing ambiguity.

Natural Function Examples

SpeakQL

*"**Get the count of** id **from the** course **table**"*

```
SQL: Select count(id) from course;
```

SpeakQL:

*"**Find the average** units **and the count of** title **in the** course **table**"*

```
SQL: Select avg(units), count(title)
     from course
```

2.4.4 Unbundling

Query unbundling provides an alternative approach to complex query formulation. In standard SQL, the select expression requires query writers and speakers to specify all columns, scalars, and functions first before then proceeding to define table sources, subqueries, and joins, followed by expressing all restrictions in the form of where predicates. Only after this process has been completed may the query speaker then modify the expression with result row limits and ordering, apply selections to aggregate values using having, and grouping non-aggregate columns using the group by clause. This requires the retention of key information within a query speaker's working memory for the duration of the query dictation process.

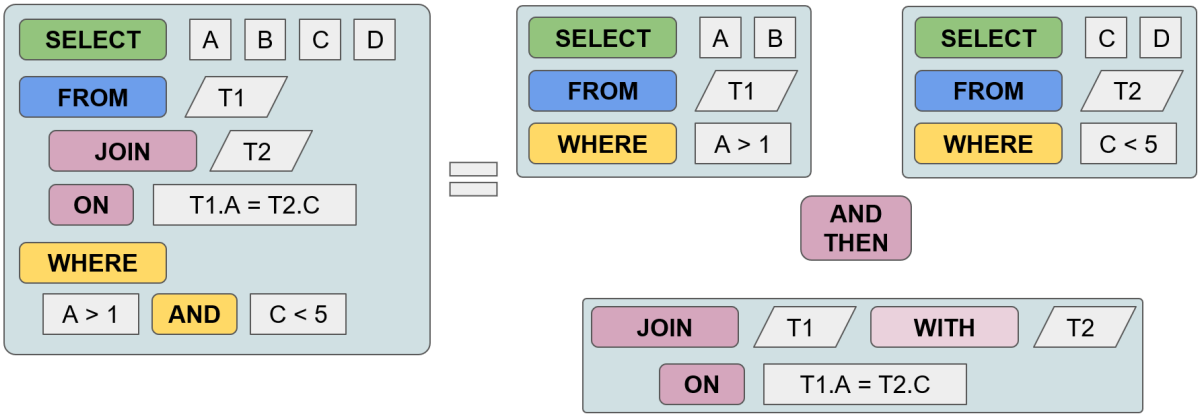


Figure 2.4. Query Unbundling

The goal of unbundling is to reduce the cognitive load on query speakers by introducing a new grammar that permits the expression of *unbundled select* subqueries that specify columns,

table source, and where predicates for a single relation at a time. The tables specified in these unbundled queries are joined together using separate *join-with* subqueries where query speakers define the relationship between two tables in their objective query. Modifier clauses may be specified together in a single expression, or seperately using multiple *modifier* subquery expressions. The order of the *unbundled select*, *join-with*, and *modifier* subqueries is optional, introducing additional flexibility to the query dictation process. Each unbundlined sub-query is delimited by the *and then* keywords.

As the following examples will demonstrate, unbundled queries tend to contain more tokens than their corresponding SQL translations and are therefore longer and likely take more time to dictate. This is the result of decisions made within the trade space between the necessary brevity of keyboard-typed SQL and the natural flow of spoken language where SpeakQL introduces optional phrases and keywords to improve the dictation process. Therefore, we sacrifice some time-based efficiency in an effort to decrease the likelihood of semantic errors and improve the users' overall query dictation experience.

Unbundling Examples

Query Prompt

What is the average room seating capacity of rooms in buildings where the course with id 'CSE 232' has ever been offered?

SpeakQL

*"**Get** buildingname **from** building*
***and then from** room **where** floor = 3 **select average** capacity*
***and then select** nothing **from** courseoffering **where** courseid = 'CSE232'*
***and then join** courseoffering **with** room **on** courseoffering.roomid = room.id*
***and then join** room **with** building **on** room.buildingid = building.id*
***and then group by** automatically"*

```
SQL: Select buildingname, avg(capacity)
```

```
from courseoffering
join room on courseoffering.roomid = room.id
join building on room.buildingid = building.id
where courseoffering.courseid = 'CSE232'
group by buildingname;
```

Query Prompt

Find the titles of all courses offered in terms with the year 2022.

SpeakQL

*"**join the** term **table with the** room **table on** term.id = courseoffering.termid*

***and then join the** courseoffering **table with the** course **table on** courseoffering.courseid = course.id*

and then show me** title **in the** course **table

***and then from** room **where** floor = 3 **select average** capacity*

***and then get** nothing **from** term **where** year = 2022"*

```
SQL: Select title from course
     join courseoffering
        on course.id = courseoffering.courseid
     join term
        on courseoffering.termid = term.id
     where year = 2022;
```

2.5 SpeakQL 2 Implementation

The objective of SpeakQL and SpeakQL 2 system implementations is to enable execution of SQL queries on arbitrary relational databases. We achieve this objective using the SpeakQL dialect by creating a SpeakQL dialect to SQL translator. This allows us to leverage existing database management systems and eliminates the requirement to create SpeakQL dialect-specific database engine optimisations.

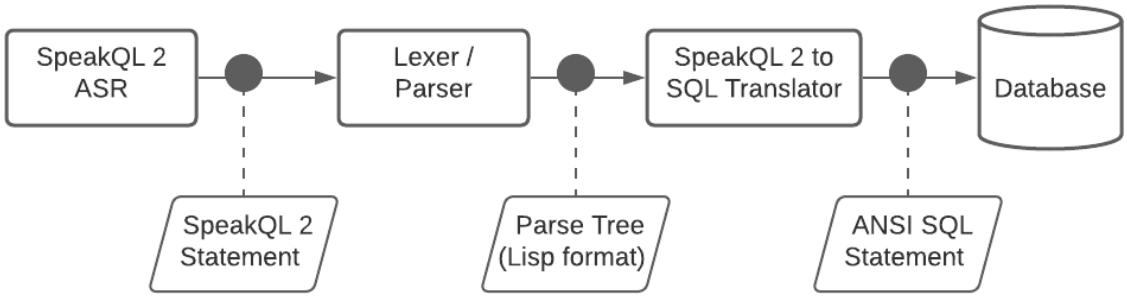


Figure 2.5. SpeakQL Dialect to ANSI SQL Translation Process

2.5.1 SpeakQL Dialect Lexer and Parser

We make use of ANTLR's implementation of ALL(*) [10], specifically the feature that enables context-based parsing using look ahead parsing to resolve potential ambiguity encountered while generating a statement's abstract syntax tree. That is, the parser recognizes its current location within the grammar and uses context to identify the parser rule associated with the next word in the query. This affords some flexibility in ordering of parser rules associated with SpeakQL features. For example, we add an intermediate rule layer between the SQL querySpecification rule and the SQL SELECT and FROM keywords, and title the associated rules in this layer selectExpression and tableExpression respectively. We can then create two alternate orderings (select then table, or table then select) both of which the parser recognizes as valid and correctly parses based on preceding and succeeding words in a given query.

2.5.2 SpeakQL Dialect Grammar

The SpeakQL 2 grammar is a modified extension of the MySQL grammar [1] described using the ANTLR language definition syntax. While the majority of the MySQL grammar remains unmodified, we make some significant adjustments within the *selectStatement* and *querySpecification* parser rules, which are rules within the dmlStatement rule set. Figures 2.6, 2.7, 2.8, 2.9, 2.10, 2.11, and 2.12 portray most, but not all, of the grammar rules modified to enable SpeakQL dialect features. Most parser rule names appear in truncated form in the

included figures, and some rules have been omitted based on their obscurity and lower likelihood of being used within a spoken query. The full grammar, including full parser rule names and rules not depicted in this paper, may be viewed in the ANTLR source files that define the full SpeakQL dialect grammar.

Synonym and Syntactic Sugar Grammars

To enable the synonym feature, we create additional keyword parser rules as intermediate rules within the SpeakQL parser that contain multiple keyword synonyms. For example, the selectKeyword parser rule contains the select synonym keywords (e.g. *(*get, display, show me) etc.). Parser rules *selKW*, *joinKW*, and *frmKW* in Figures 2.7, and 2.8 exist one level above terminal representative of each SpeakQL synonym. These keyword-based intermediate parser rules provide the SpeakQL-to-SQL translator with a reference point for keyword replacement as depicted in algorithm 1.

Alternate Ordering Grammars

Parser rule queryOrdSpec (query order specification) in Figure 2.7 contains four branches representing the four alternate ordering options for the select statement's *select*, *from-join*, and *where* expressions. The selModExpr (select modifier expression) also in Figure 2.7 contains a single branch that contains four optional rules that serve as placeholders for any of the four modifier clauses *group by*, *having*, *order by*, and *limit*. This approach differs from the select expression branching due to the increased number of ordering options. Because each placeholder rule can contain any of the modifier clauses, the SpeakQL-to-SQL translater assumes the responsibility for addressing cases where a query speaker incorrectly specifies more than one limit clause or conflicting order-by clauses.

Natural Function Grammars

The SpeakQL grammar retains the *functionCall* parser rule in the original MySQL grammar from which it was derived, and adds an additional child rule as a sibling to the *aggregateWin-*

dowedFunction parser rule that specifies functions commonly used to fulfill analysis-focused tasks (e.g. *count*, *sum*, etc.). The additional sibling rule, titled *noParenAggregateWindowedFunction*, contains a subset of the functions available in the *aggregateWindowedFunction* parser rule. Additionally, as depicted in figure 2.11, the *functionArg* parser rule which enables a wide range of rules to exist as parameters, is replaced by a more restrictive rule that allows only *constant* and *columnName* rules to exist as legal children rules that follow the name of the function. This constraint mitigates unintended ambiguity may occur when attempting to specify a complex expression without surrounding parentheses.

Unbundling Grammars

Our approach to defining unbundling grammar involves the creation of additional rules (see figure 2.12) that delimit between unbundled expressions (*expressionDelimiter*), restrict the use of joins within the from clause (*fromClauseNoJoin*), and define a standalone join expression (*multiJoinExpression*) that employs the *withKeyword* rule to specify a join relationship between two tables or subqueries. Unbundled grammar is accessed at a relatively high level in the SpeakQL grammar from the (*querySpecification* parser rule (see figure 2.6).

Generally, the parser will recognize that a specific query is employing unbundling features when the query speaker specifies an *expressionDelimiter* token (e.g. *and then, then, or next*) between two unbundled subqueries. Once this occurs, the parser will only recognize queries as legal if they do not contain join expressions within the table expression, as all joins in unbundled queries must be defined using the *multiJoinExpression* parser rules. Although not expected to occur often, unbundled queries may contain subqueries in lieu of table names. These subqueries do not necessarily have to conform to the unbundled syntax, and may follow either the unbundled or non-unbundled paths through the grammar.

¡selStmt¿ ::=	¡qrySpec¿
¡qrySpec¿ ::=	¡qryOrdSpec¿ ¡selModExpr¿ — (¡multiJoinExpr exprDelim)? (¡selModExpr¿ ¡exprDelim¿)? ¡ubndQryOrdSpc¿ (¡exprDelim¿ (¡ubndQryOrdSpc¿ — ¡multiJoinExpr)) (¡exprDelim¿ ¡selModExpr)?
¡qryOrdSpec¿ ::=	¡selExpr¿ ¡whereExpr¿? ¡tabExpr¿ — ¡selExpr¿ ¡tabExpr¿ ¡whereExpr¿? — ¡tabExpr¿ ¡selExpr¿ ¡whereExpr¿? — ¡tabExpr¿ ¡whereExpr¿? ¡selExpr¿
¡ubndQryOrdSpc¿ ::=	¡selExpr¿ ¡whereExpr¿? ¡tabExprNoJoin¿ — ¡selExpr¿ ¡tabExprNoJoin¿ ¡whereExpr¿? — ¡tabExprNoJoin¿ ¡selExpr¿ ¡whereExpr¿? — ¡tabExprNoJoin¿ ¡whereExpr¿? ¡selExpr¿

Figure 2.6. Query Expression Grammar

¡selExpr¿ ::=	¡selKW¿ ¡selSpec¿* ¡selElmts¿ — ¡selKW¿ ¡nothingElmt¿
¡selKW¿ ::=	”select” — ”find” — ”retrieve” — ”get” — ”show me” — ”display” — ”present” — ”what is” — ”what is the” — ”what are” — ”what are the”
¡selSpec¿ ::=	”all” — ”distinct” — ”distinctrow
¡selElmts¿ ::=	(”*” — ¡selElmt¿) (¡delim¿ ¡selElmt¿)*
¡delim¿ ::=	”,” — ”and”
¡selElmt¿ ::=	”*” — ¡colName¿ — ¡funCall¿
¡nothingElmt¿ ::=	¡nothingKW¿
¡nothingKW¿ ::=	”nothing”

Figure 2.7. Select Expression Grammar

¡tabExpr¿ ::=	¡frmKW¿ ¡tblSrcs¿
¡frmKW¿ ::=	”from” — ”from table” — ”from tables” — ”in table” — ”in tables”
¡tblSrcs¿ ::=	¡theKW¿? ¡tblSrc¿ ¡tabKW¿? (¡delim¿ ¡theKW¿? ¡tblSrc¿ ¡tblKW¿?)*
¡tblSrc¿ ::=	¡tblSrcItm¿ ¡joinPart¿* — ”(” ¡tblSrcItm¿ ¡joinPart¿ ”)”
¡tblSrcItm¿ ::=	”(” ¡slctStmt¿ ”)” — ¡tblName¿ ¡tblAlias¿? — ”(” ¡tblSrcs¿ ”)”
¡joinPart¿ ::=	¡inJoin¿ — ¡outJoin¿ — ¡natJoin¿
¡inJoin¿ ::=	¡inJoinKW¿? ¡joinKW¿ ¡tblSrcItm¿ (¡onKW¿ ¡expr¿)
¡inJoinKW¿ ::=	”inner” — ”cross”
¡outJoin¿ ::=	¡joinDir¿ ¡outJoinKW¿? ¡joinKW¿ ¡tblSrcItm¿ (¡onKW¿ ¡expr¿)
¡joinDir¿ ::=	”left” — right
¡outJoinKW¿ ::=	”outer”
¡natJoin¿ ::=	¡natJoinKW¿ (¡joinDir¿ ¡outJoinKW¿)? ¡joinKW¿ ¡tblSrcItm¿
¡natJoinKW¿ ::=	”natural”
¡joinKW¿ ::=	”join” — ”join table” — ”by joining” — ”by joining table” — ”joined with” — ”join with” — ”joined with table” — ”join with table” — ”by joining with table”
¡onKW¿ ::=	”on”
¡theKW¿ ::=	”the”
¡tblKW¿ ::=	”table”

Figure 2.8. Table Expression Grammar

¡whereExpr¿ ::=	¡whereKW¿ ¡expr¿
¡whereKW¿ ::=	”where”
¡expr¿ ::=	(”not” — ”!”) ¡expr¿ — ¡expr¿ ¡logicOp¿ ¡expr¿ — ¡predicate¿ ”is” ”not”? (”true” — ”false” — ”unknown”) — ¡predicate¿
¡logicOp¿ ::=	”and” — ”xor” — ”or”
¡predicate¿ ::=	¡predicate¿ ”not”? ”is” ”in” ¡leftParen¿ ¡selStmt¿ ¡rightParen¿ — ¡predicate¿ ”is” ”not” ”null” — ¡predicate¿ ¡compareOp¿ ¡predicate¿ — ¡predicate¿ ¡compareOp¿ (”all” — ”any” — ”some”) ¡leftParen¿ ¡selStmt¿ ¡rightParen¿ — ¡predicate¿ ”not”? ”between” ¡predicate¿ ”and” ¡predicate¿ — ¡predicate¿ ”not”? ”like” ¡predicate¿ — (expressionAtom)

Figure 2.9. Where Expression Grammar

¡selModExpr¿ ::=	¡selModItm¿? ¡selModItm¿? ¡selModItm¿? ¡selModItm¿?
¡selModItm¿ ::=	¡grpByCls¿ — ¡havingCls¿ — ¡orderByCls¿ — ¡limitCls¿
¡grpByCls¿ ::=	¡grpByKW¿ ¡grpByItm¿ (¡delim¿ ¡grpByItm¿)* (”with” ”rollup”)?
¡grpByKW¿ ::=	”group by” — ”group”
¡grpByItm¿ ::=	¡grpByExpr¿ ¡order¿ — ¡autoMtcKW¿
¡autoMtcKW¿ ::=	”automatic” — ”automatically”
¡grpByExpr¿ ::=	”not” ¡grpByExpr¿ — ¡predicate¿ ”is” ”not”? (”true” — ”false” — ”unknown”) — ”predicate”
¡delim¿ ::=	”,” — ”and”
¡havingCls¿ ::=	¡havingKW¿ ¡expr¿
¡havingKW¿ ::=	”having”
¡ordrByCls¿ ::=	”order by” ¡ordrByExpr¿ (¡delim¿ ¡ordrByExpr¿)
¡ordrByExpr¿ ::=	¡expr¿ ¡order¿?
¡order¿ ::=	¡ascKW¿ — ¡descKW¿
¡ascKW¿ ::=	”asc” — ”ascending”
¡descKW¿ ::=	’desc’ — ’descending’
¡limitCls¿ ::=	’limit’ ¡limitClsAtm¿
¡limitClsAtm¿ ::=	(decimalLiteral — simpleId)

Figure 2.10. Select Modifiers Expression Grammar

¡funCall¿ ::=	¡aggrFun¿ — ¡noParAggrFun¿ — ¡nonAggrWinFun¿ — ¡sclrFunNam¿
¡aggrFun¿ ::=	¡theKW¿? (”avg” — ”average” — ”max” — ”min” — ”sum”) ¡ofKW¿? ”(” (”all” — ”distinct”)? **¡funArg¿** ”)” — ¡theKW¿? ”count” ¡ofKW¿? ”(” (”*” — ”all”? **¡funArg¿** — ”distinct” **¡funArg¿**) ”)” (”std” — ”std dev” — ”std dev pop” — ”std dev samp” — ”var pop” — ”var sample” — ”variance”) ”(” ”all”? **¡funArg¿** ”)” ”group concat” ”(” ”distinct”? **¡funArg¿** (”order by” ¡ordByExp¿ (¡delim¿ ¡ordByExp¿)*) ”)”
¡funArg¿ ::=	(¡constant¿ — ¡colName¿ — ¡funCall¿ — ¡expr¿) (¡delim¿ (¡constant¿ — ¡colName¿ — ¡funCall¿ — ¡expr¿))*
¡noParAggrFun¿ ::=	¡theKW¿? (”avg” — ”average” — ”max” — ”min” — ”sum”) ¡ofKW¿? (”all” — ”distinct”)? **(¡constant¿ — ¡colName¿)** — ¡theKW¿? ”count” ¡ofKW¿? (”*” — ”all”? ¡funArg¿ — ”distinct” **(¡constant¿ — ¡colName¿)**)

Figure 2.11. Function Call Grammar

¡tabExprNoJoin¿ ::=	¡frmClsNoJoin¿
¡frmClsNoJoin¿ ::=	¡frmKW¿ ¡theKW¿? ¡tblSrcNoJoin¿ ¡tblKW¿?
¡tblSrcNoJoin¿ ::=	¡tblSrcItm¿ — ”(” ¡tblSrcItm¿ ”)”
¡mltiJnExpr¿ ::=	¡mltiJnPrt¿ (¡exprDelim¿ ¡mltiJnPrt¿)*
¡mltiJnPrt¿ ::=	¡mltiInnrJn¿ — ¡mltiOutJn¿ — ¡mltiNatJn¿
¡exprDelim¿ ::=	”and then” — ”then” — ”next”
¡mltiInnrJn¿ ::=	¡inJoinKW¿? ¡joinKW¿ ¡tblSrcItm¿ ¡withKW¿ ¡tblSrcItm¿ (¡onKW¿ ¡expr¿ — ”using” ”(” ¡uidLst¿ ”)”)?
¡mltiOutJn¿ ::=	¡joinDir¿ ¡outJoinKW¿? ¡joinKW¿ ¡tblSrcItm¿ ¡withKW¿ ¡tblSrcItm¿ (¡onKW¿ ¡expr¿ — ”using” ”(” ¡uidLst¿ ”)”)
¡mltiNatJn¿ ::=	¡natJoinKW¿ (¡joinDir¿ ¡outJoinKW¿)? ¡joinKW¿ ¡tblSrcItm¿ ¡withKW¿ ¡tblSrcItm¿
¡withKW¿ ::=	”with” — ”with table” — ”and”

Figure 2.12. Unbundling-Related Grammar Rules

¡selStmt¿ ::=	¡qrySpec¿
¡qrySpec¿ ::=	”select” ¡selSpec¿* ¡selElmts¿ ¡fromCls¿? ¡grpByCls¿? ¡havingCls¿? ¡ordrByCls¿? ¡limitCls¿?
¡fromCls¿ ::=	(”from” ¡tblSources¿?) (”where” ¡expression¿)?
¡grpByCls¿ ::=	”group by” ¡grpByItm¿ (”,” ¡grpByItm¿)?
¡havingCls¿ ::=	”having” ¡expression¿
¡ordrByCls¿ ::=	”order by” ¡ordrByExpr¿ (”,” ordrByExpr)
¡ordrByExpr¿ ::=	¡expression¿ (”asc” — ”desc”)
¡limitCls¿ ::=	”limit” (¡limitClsAtom¿ ”,”)? ¡limitClsAtom¿ — ”limit” ¡limitClsAtom¿ ”offset” ¡limitClsAtom¿

Figure 2.13. SQL Select Statement Grammar Excerpt

2.5.3 SpeakQL Query to SQL Query Translation

The SpeakQL to SQL translation process begins with a parse operation on a valid query expressed in the SpeakQL dialect. For purposes of translation, we assume that an automatic speech recognition (ASR) and error correcting application has resolved any query dictation or speech recognition errors that may have occurred. In other words, the SpeakQL translator assumes that the tokens present in the query reflect the intent of the query writer.

The result of the query parsing operation is an abstract syntax tree (AST) containing the grammar rules associated with the statements present within the query. The AST is initially generated using a Java-based SpeakQL parser specified using the ANTLR [10] language. The Java-based result is passed to the Python-based translator as a Lisp-style tree. On receipt of the lisp tree, the translation begins when the translator parses the lisp tree and generates a Python-based AST.

Upon initialization, the Python-based AST performs cataloguing operations including cataloging and masking subqueries, generating a dictionary of referenced tables and aliases, generating a list of referenced columns, and generating a two way table-column dictionary. Tree nodes are indexed in a dictionary-based data structure, and nodes-to-node associations exist in the translator as index references enforced by the tree node index. During initialization, the translator also generates statistics useful for determining if specific translation features need to

be executed on a given query.

After AST initialization is complete, the translator begins the translation process. The result of this process is a re-ordered AST that, when serialized, outputs a valid SQL query. This is accomplished through a series of steps: replacing SpeakQL synonyms with equivalent SQL keywords, re-ordering expressions into valid SQL expression order, removing syntactic sugar, and bundling unbundled queries. These operations are performed recursively on any subqueries cataloged during initialization.

Synonym Replacement

Synonym replacement, where SpeakQL keywords are replaced with equivalent SQL keywords, occurs as the first step of query translation. To accomplish this task, the translator performs a scan search of all active nodes in the AST for rules that are a member of the keyword and delimiter categories for which SpeakQL synonyms exist. This includes *selectKeyword*, *fromKeyword*, *selectElementDelimiter*, and others. When a keyword or delimiter rule is identified during the search, the translator performs an update on the rule node which replaces the terminal token representative of a SpeakQL synonym with its corresponding SQL keyword.

```
SpeakQL rule:      'selectKeyword SHOW ME'
After replaceKW:   'selectKeyword SELECT'
```

Algorithm 1 provides a pseudo-code representation of the synonym replacement process performed by the translator on the SpeakQL AST. After completion, all SpeakQL synonym keywords that existed in the initial query and its subqueries have been replaced with valid SQL syntax.

Expression Reordering

The next step of the translation process is expression reordering. This step includes both the reordering of *select, from, where* and *group by, order by, having, limit* expressions. For this feature, we take advantage of the grammar structure and associated AST structures that

Algorithm 1. Find and Replace Synonyms

```
speakQlAst ← parseSpeakQL(query)
synonymKwRuleset ← {selectKW, fromKW, joinKW, ...}
syntaxSugarRs ← {theKW, tableKW, ofKW, isKW}
synonymHashMap ← {(selectKW : SELECT), (fromKW : FROM), ...}
for all nodes such that node ∈ speakQlAst do
  if node.rule ∈ synonymKwRuleset then
    node.replaceKW(synonymHashMap[node.rule])
  end if
  if node.rule ∈ syntaxSugarRs then
    speakQlAst.removeNode(node)
  end if
end for
```

guarantees that all children of the *queryOrderSpecification*, *unbundledQueryOrderSpecification*, and *selectModifierExpression*, parser rule nodes are rules that should be evaluated and reordered. Specifically, we can be certain that the query order expression parser rules will only contain the children *selectExpression*, *whereExpression*, *tableExpression*. We are also certain that the *selectModifierExpression* has no more than four children of type *selectModifierItem*, each of which may have only one child of type *groupByClause*, *havingClause*, *orderByClause*, or *limitClause*. These guarantees enable us to limit the scope of reordering operations to only the children of the parent nodes of the sub-expressions.

Algorithm 2 portrays the process by which the translator modifies the AST. Expression reordering is accomplished by iteration over the array containing the children of *queryOrder-Specification*. When the first rule in the SQL order (*selectExpression*) is encountered, this node is added to a new children list object. Iteration over the remaining children rules continues until all children have been identified and reordered into the new children list in SQL-correct order. The translator then updates the parser rule node *queryOrderSpecification*, replacing its old children list with a new list containing the same children, but in SQL-correct order.

Algorithm 3 portrays a similar process that reorders what we refer to the select modifier rules which includes *groupByClause*, *havingClause*, *orderByClause*, and *limitClause* parser rules. The translator employs the same strategy of iteration over the parent rule's children, except

Algorithm 2. Reorder Select, From and Where Expressions

```
speakQlAst ← parseSpeakQL(query)
for all ruleNodes such that ruleNode ∈ speakQlAst do
  if ruleNode == qryOrdSpec then
    expr ← ruleNode.children
    reorderedExpr ← [∅]
    reorderedExpr.append(expr.selExpr)
    reorderedExpr.append(expr.tabExpr)
    reorderedExpr.append(expr.whereExpr)
    ruleNode.children ← reorderedExpr
  end if
end for
```

that it must descend one level deeper through the additional intermediate layer consisting of up to four *selectModifierItem* rules that exist between the parent *selectModifierExpression* and child clause parser rules. The translator iterates over the original child list and places the clauses in SQL-correct order into a new list. After iteration is complete, the *selectModifierExpression* children are replaced with the SQL-ordered list of select modifier clauses.

Algorithm 3. Reorder Select Modifier Items

```
speakQlAst ← parseSpeakQL(query)
for all ruleNodes such that ruleNode ∈ speakQlAst do
  if ruleNode == selModExpr then
    reorderedItms ← [∅]
    grpByCls ← ruleNode.getChildByRule(grpByCls)
    havingCls ← ruleNode.getChildByRule(hvingCls)
    ordrByCls ← ruleNode.getChildByRule(ordrByCls)
    limitCls ← ruleNode.getChildByRule(limitCls)
    reorderedItms.append(grpByCls)
    reorderedItms.append(havingCls)
    reorderedItms.append(ordrByCls)
    reorderedItms.append(limitCls)
    selModExpr.children ← reorderedItms
  end if
end for
```

Natural Function Transformation

The SpeakQL natural functions feature allows for the omission of parentheses and the inclusion of optional *the* and *of* keywords for functions where only a single column or constant value is passed as argument. Translating natural functions to SQL-correct function statements involves the removal of the optional keyword parser rules and the insertion of parentheses. The translator's AST class contains helper functions *removeNodeFromTree* and *surroundNodeWithParens* used to accomplish these tasks.

Query Bundling

Query bundling is the series of tasks the translator performs to combine a set of unbundled queries into a valid SQL statement.

Algorithm 4. Transform Unbundled Query (High Level)

$speakQlAst \leftarrow parseSpeakQL(query)$
Require: $\exists unbndQryOrdSpc \in speakQlAst$
$speakQlAst.inferGroupBy()$
$speakQlAst.bundleSelectElements()$
$speakQlAst.bundleWhereStatements()$
$speakQlAst.bundleJoinParts()$
$speakQlAst.bundleTables()$
for all *ruleNodes* such that $ruleNode \in speakQlAst$ **do**
  if $ruleNode \in \{qryOrdSpec, multQryOrdSpec, exprDelim\}$ **then**
    $speakQlAst.removeNode(ruleNode)$
  end if
end for

At a high level, this is accomplished through a series of up to five steps (see algorithm 4). The actual number of steps executed for a given query depends upon the existence of certain statements within the query. For example, the *inferGroupBy* step will be skipped for queries that do not contain aggregator functions and an associated group by expression. However, every valid SpeakQL query that employs unbundling will, at a minimum, execute the *bundleSelectElements*, and either *bundleJoinParts* or *bundleTables* methods. Unbundling steps are performed in the order:

1. Infer group by expression (optional)
2. Bundle select elements
3. Bundle where statements (optional)
4. Bundle join parts (optional)
5. Bundle tables (optional)

The translator generally uses the same approach for each bundling step, which is to identify the first expression parser rule node in the AST for a given expression and use it as a migration target for additional expression parser rule nodes that may exist in the AST. For example, if a query contains two separate unbundled select-from-where queries connected with a join query, the translator will designate the first of the two *selectExpression* parser rules in the AST as *selectExpression'* and will migrate the *selectElements* that are children of the second *selectExpression* rule node to become children of *selectExpression'*. Additional considerations apply individually to each bundling step, and will be discussed in more detail in the step-specific paragraphs that follow. Parser rule node migration is accomplished by appending the migrated parser rule node's ID to the migration target rule node's child list and removing the migrated parser rule node's ID from the migration source rule node's child list.

Inferring the group by expression

A feature of the SpeakQL translator is its ability to infer a group by expression using the presence of aggregator functions and column references within multiple *selectExpression* parser rules. A user can instruct the translator to perform group by expression inference using the *group by automatically* keyword within either a single-table SpeakQL query or a multi-table unbundled query.

When the translator encounters the *group by automatically* instruction, it proceeds as depicted in algorithm 5. It makes use of AST helper functions *nodeWithName* which returns a

Algorithm 5. Unbundled Function: Infer GroupBy

```
speakQlAst ← parseSpeakQL(query)
Require: ∃grpByCls ∈ speakQlAst ∧ ∃autoMtcKW ∈ speakQlAst
autoKws ← speakQlAst.nodesWithName(autoMtcKW)
for all autoKw ∈ autoKws do
    speakQlAst.removeNode(autoKw)
end for
gbNode ← speakQlAst.nodeWithName(grpByCls)
elmtsByTbl ← speakQlAst.getAllTablesAndElements()
gbItms ← [∅]
for all selElmt ∈ elmtsBtTbl do
    if selElmt is column then
        gbItems.append(selElmt)
        gbItems.append(delim)
    end if
end for
gbNode.children ← gbItms
```

list of all nodes with a given rule name, and *getAllTablesAndElements* which returns a Python dictionary ({ *table name* : [*selectElements*]}) that contains a list of select elements and function calls for each table referenced within the SpeakQL query. The translator uses the existing *groupByExpression* parser rule node that contains the *auomaticGroupByKeyword* parser rule and replaces the *automaticGroupbyKeyword* rule with a *groupByItems* parse rule. The *groupByItems* parse rule node serves as the parent node for individual *groupByItem* and *groupByItemDelimiter* nodes generated from non-function *selectElement* nodes registered in the table-element python dictionary. Because the *groupByItems* parse rule already exists beneath the *selectModifierExpression* parse rule, no further AST transformations are required; and the group by inference operation is complete after the generation of the *groupByItems* parse rule and its children *groupByItem* and *groupByItemDelimiter* parse rules.

Bundling select elements

Select element bundling is a required step for any SpeakQL query that makes use of the unbundling feature. Because any SpeakQL query that uses unbundling must have at least two *selectExpression* parse rules, select element bundling is a mandatory step of the bundling

process.

Algorithm 6. Unbundled Function: bundleSelectElements

```
speakQlAst ← parseSpeakQL(query)
for all ruleNode ∈ speakQlAst.nodesWithName(nothingElmt) do
    ruleNode.rename(selElmt)
end for
elmtsByTbl ← speakQlAst.getAllTablesAndElements()
selElmtsRlNd′ ← speakQlAst.nodesWithName(selElmts)[0]
newChildren ← [∅]
for all tblItm ∈ elmtsByTbl do
    for all selectElmt ∈ elmtsByTbl[tblItm] do
        newChildren.append(tblItm.selectElmt)
    end for
end for
selElmtsRlNd′.children ← newChildren
for all ruleNode ∈ speakQlAst.nodesWithName(selElmts) such that ruleNode! =
selElmtsRlNd′ do
    speakQlAst.removeNode(ruleNode)
end for
```

As shown in algorithm 6 as *selElmtsRlNd'*, which abbreviates *selectElementsRuleNode*, the translator uses the first *selectElements* parser rule as a migration target for all other select elements within the SpeakQL query. Also depicted in the same algorithm is a preemptive method for avoiding ambiguity where the translator prepends each migrated select element with its associated table, resulting in a dotted id in the format *tableName.columnName*. This step is performed for both standalone column references and column references as arguments within function expressions. After iterating through each *selectElementExpression* parser rule and migrating their associated *selectElements* children to *selectElementsRuleNode'*, a cleanup operation removes them from the AST leaving a single *selectElementsExpression* in the AST that contains all column and function references present in the original SpeakQL unbundled query.

Bundling where expressions

The where expression bundling step is optional, and is only invoked if at least one bundled query expression contains a *whereExpression* parse rule. When performing where expression

bundling, the SpeakQL translator makes the following assumptions:

- All cross-table predicates are conjunctive (and)
- Multiple predicate expressions within a single unbundled query should be encased in parentheses before consolidation

Given these assumptions, where expression bundling proceeds as depicted in algorithm 7.

Algorithm 7. Unbundled Function: bundleWhereStatements

$speakQlAst \leftarrow parseSpeakQL(query)$
Require: $\exists whereExpr \in speakQlAst$
$unbdlExprs \leftarrow speakQlAst.nodesWithName(unbdlQryOrdSpc)$
$unbdlExpr' \leftarrow unbdlExprs[0]$
$whereExpr' \leftarrow unbdlExpr'.nodeWithName(whereExpr)$
for all $exprNode \in unbdlExprs[1:]$ **do**
  $tblName \leftarrow exprNode.nodeWithName(tblName)$
  $exprWhereExpr \leftarrow exprNode.nodeWithName(whereExpr)$
  $exprWhereKw \leftarrow exprWhereExpr.nodeWithName(whereKW)$
  $exprWhereExpr.removeNode(exprWhereKw)$
end for

Similar to the select element bundling step, the first occurence of a *whereExpression* parse rule, *whereExpression'*, within the query AST and designates it as the target for additional *whereExpression* parse rule migration. Because where expressions are optional within *selectStatement* parser rules, it is possible that *whereExpression'* is not a member of the first *selectStatement* parse rule, *selectStatement'*, in the query. Because of this possibility, the translator may perform an additional migration step where it migrates *whereExpression'* to become a child of *selectStatement'*.

The translator takes advantage of the recursive nature of the where expression grammar (see figure 2.9) by designating where expressions that exist in *selectStatement* parse rules that are not *selectStatement'* as children of *whereExpression'*. During this migration process, the translator employs helper functions to surround migrated expressions with parentheses and

deliminating the migrated expressions with *andKeyword* parser rules. The end result of this process is a single cross-table conjunctive *whereExpression* parser rule that contains all other *whereExpression* parser rules specified in other unbundled queries within the SpeakQL query.

Bundling join parts

Join part bundling collects all *multiJoinExpression* parse rules and aggregates them in an arbitrary order to form a valid SQL join expression. Join part bundling is an optional feature because an alternate form of relation joining where all table sources are specified within *fromExpression* parse rules, and join conditions between each table source are specified within *whereExpression* predicates (e.g. *from table one, table two where one.id = two.id*), is also possible. Currently, because join order in the resulting SQL query is arbitrary within the bounds of its implementation rules, join part bundling is limited to allowing only inner joins. Queries that require outer join expressions may still be expressed using other SpeakQL features; but may not employ the unbundling feature. Outer join capability is a future SpeakQL feature development objective.

The join part bundling process begins when the translator creates a list of all *multiJoinExpession* parse rule nodes within the AST. Given this list, it performs iterative analysis on each expression to determine if a subquery exists as a table item within any of the *multiJoinExpression* nodes, and if so, substitutes the subquery with a subquery masking rule and alias in the join expression. The translator then checks the left and right table reference parse rules within the join expression to determine if an alias exists for each table, or if the table is referenced in the join expression by its alias. If an alias association is encountered, it creates *asKeyword* and *multiJoinTableAlias* parse rules and adds them as children to the target *joinExpression* parser rule that represents the objective SQL-correct join expression.

After analyzing all *multiJoinExpression* and making modifications toward SQL-correct syntax, the translator begins the join consolidation process by designating the first table referenced in the queries first *selectStatement'* as the base table upon which the SQL-correct join statement

will be built. In order to ensure valid join expression chaining, the translator determines a table join order that ensures that each subsequent table added to a join expression references a table in its join condition that already exists within the objective join expression. In other words, while building the SQL-correct join expression, a *multiJoinExpression* will not be added to the target *joinExpression* until all tables referenced in its join condition have been added to the target *joinExpression*. The process will continue iterating over remaining *multiJoinExpressions* until the table existence constraint can be satisfied and all expressions have been added to the objective SQL-correct expression, or it is determined that *multiJoinExpressions* exist in the SpeakQL query that cannot be chained and the translator responds with an error condition. The translator then performs an additional safety check to ensure that all tables referenced in the query's *unbundledQueryOrderSpecification* parser rules are also referenced in a corresponding *multiJoinExpression* parser rule, and throws an error if a violation is encountered.

After the join bundling process is completed, all join expressions consolidated within a single *multiJoinExpression* are migrated to the first *tableExpressionNoJoin* parse rule. The *tableExpressionNoJoin* parse rule name is then updated to *tableExpression* and becomes a valid SQL from + join expression; and join bundling is complete.

Bundling table expressions

The final step of the bundling process involves collection of all *tableExpressionNoJoin* parse rule nodes within the AST. This is only required in cases where an unbundled query contains multiple table references but has no associated *multiJoinExpression* parser rules. In this case the first *tableExpressionNoJoin*, designated as *tableExpressionNoJoin'*, is established as the expression migration target, and additional *tableExpressionNoJoin* rules that exist as children of additional *selectStatement* rules are added to *tableExpressionNoJoin'*. If a where expression exists in any *(*selectStatement) that defines a join condition between two tables within the query, the expression consolidation occurs during the previously described where expression bundling step. If no where condition is defined between two tables within the query, the translator still

performs table expression bundling, and the result is a SQL statement that produces a cartesian product of the two tables.

Syntax tree cleanup

After all relevant bundling steps are complete, the translator performs syntax tree cleanup by removing parse rules from which their child sub expressions have been migrated. Upon completion of tree cleanup, the SpeakQL to SQL translation process is complete, and serialization of the abstract syntax tree's terminal nodes results in a valid SQL statement equivalent to the input SpeakQL query.

2.6 SpeakQL 2 Dialect Evaluation

We are currently implementing a mixed-methods, within-subjects, comparative usability study of the SpeakQL 2 prototype language features by recruiting participants with SQL experience to spend approximately 90 minutes watching an instructional video and answering 12 questions using both SQL and SpeakQL 2. Our objective was to identify which, if any, SpeakQL 2 dialect features made the process of dictating queries easier and / or reduced the likelihood of errors.

2.6.1 Objectives

Research Questions

Our questions related to the structure and content of the proposed SpeakQL 2 dialect. Given the preliminary features for the SpeakQL dialect, and we sought to determine which of them actually improved the query dictation process.

Q1 - Effectiveness of alternate syntax:

To what extent, if any, does introducing syntax synonyms improve the user experience of constructing a query using speech recognition software?

Q2 - Effectiveness of alternate structure:

To what extent, if any, does altering the structure and order of SQL statements reduce the tendency for errors when constructing a query using speech recognition software?

Q3 - Effectiveness of unbundling:

To what extent, if any, does unbundling a complex query into smaller relation-focused parts reduce the possibility of failing to produce a valid query using speech recognition software?

Hypotheses

H1 - Synonyms and Reduction in Special Characters Improve User Experience

Introducing syntax synonyms and removing the requirement to speak special characters like 'comma', 'parenthesis', or 'asterisk', makes speaking a query feel more natural and improves user experience and reduces the number of errors and time taken to form simple queries.

H2 - Alternate Ordering Reduces Errors

Relaxing the ordering requirement for various subexpressions reduces the likelihood of ordering-based syntax errors and thus reduces the amount of time and number of attempts required to dictate a correct simple or complex query.

H3 - Unbundling Unburdens Working Memory

As the number of tables and corresponding column references in a query increases, so does the probability of error when speaking a query. Breaking a complex, multi-table query into a series of individual single-table queries reduces the query speaker's working memory burden and reduces the likelihood of errors when constructing a query using speech recognition software. This should reduce the number of attempts required to form a correct complex query containing multiple relations.

2.6.2 Protocol

Participants

We have currently conducted 17 study sessions, 11 of which were with participants who were graduate students in data or computer science, one undergraduate computer science student, two senior professionals, and one junior professional. Two participants opted not to disclose any demographic or career information. We primarily targeted university programs that focus on analytics, and computer and data science. Because the SpeakQL dialect is an extension of the SQL syntax, we sought out participants who were already experienced with SQL which reduced the time required for training. Prospective participants were asked to complete a short SQL skills assessment, those who passed the screening questionnaire were then invited to participate in the experiment.

Study Design

Context

As both a COVID safety measure, and in an effort to recruit from as broad a pool as possible, the entire study is being performed online using a video conferencing program and a custom-built browser-based user interface. The user interface enables participants to view a database schema, receive query prompts, dictate their queries through their device's microphone, and review a speech-to-text transcription of their dictation. The research facilitator uses a seperate 'wizard of oz' study control panel to evaluate user performance in terms of query correctness, provide feedback, answer SQL- and SpeakQL-related questions, and manage the progression of the study.

Quantitative Analysis

We evaluate performance using:

- Total-Time-to-Completion: Lower is better. Total time (in seconds) from when the participant is presented with the question prompt to when they press the submit button for

their final query submission.

- First Attempt Planning Time: Lower is better. Time (in seconds) from when the participant is presented with the question for the first time to when they press the record button.
- Dictation (recording) Time: Lower is better. Time (in seconds) from when the participant presses the record button to when they press the stop button.
- Number of Attempts: Lower is better. Discrete (1, 2, 3) inicating how many attempts the participant made to form a correct query.

We measure these variables both in terms of overall averages (mean) within each set, as well as percent change between dialects for each SpeakQL-SQL response pair used to answer a question. The latter approach mitigated the increased variance caused by the intermixing of simple and complex queries within the question set.

- Query Complexity: Continuous positive real number derived from the presence and quantity of columns, tables, joins, functions and predicates.
- Query is Complex: Discrete (0, 1) simpler method for indicating if a query is complex. Based on complexity threshold defined below.
- Query Attributes: Number of projections, number of tables, number of selections, number of joins, and number of modifiers required to answer the question.
- Features Used: One-hot encoded vector representing the SpeakQL features actually used by a participant for a given SpeakQL query.

Determining Feature Usage

We analyze feature usage in each query attempt by performing speech-to-text transcription on attempt recordings. The speech-to-text service was provided by an Amazon Web Services Transcribe custom language model trained on 500,000 procedurally generated SpeakQL and SQL queries against the same database used in the study.

Measuring Query Complexity

Query complexity is determined using a series of weighted criteria that include the number of tables (relations) in a query, the number of projections (columns, functions and constants within the select expression), number of functions, number of selections (where expressions), number of joins, and number of modifiers (group by, having, order by and limit). A query is considered to be complex if the standardized sum of the weighted criteria for a query is greater than 0.

$$Complexity = Standardize(\sum_{i=1}^{n} Weight_i * Score_i)$$

Where *Weight* is a vector length *n* of weights associated with values in vector *Score* of length *n* and the standardization function computes a value between -1 and 1 using the complexity score of a query minus the mean divided by the standard deviation of all query complexity scores.

$$Standardize(WeightedScore) = \frac{WeightedScore - \mu_{Scores}}{\sigma_{Scores}}$$

Data Collection

Data generated by the study includes time to plan (time from updating the query prompt until pressing record), time to record (time from pressing record to stopping the recording), number of attempts per query, dictation transcript, and audio recordings of each dication attempt. All data is captured within participants' browser sessions and uploaded to a back end web server and database for processing and analysis during the session. In an effort to add explanatory power to our quantitative results, we ask participants to complete a voluntary post-participation survey.

Managing the Learning Effect

We applied a latin squares approach using counterbalancing to counteract the learning effect [9] inherent in within-subjects studies of two or more treatments. Specifically, participants

were divided into two groups: a SpeakQL-to-SQL group and a SQL-to-SpeakQL group, where participants in one group answered all 12 questions in one dialect and then switched to the opposite dialect and answered the same 12 questions again. Post-participation analysis confirmed the efficacy of this control by verifying that little-to-no assymetric learning occurred between dialects.

2.6.3 Current Results

So far we have collected 213 observations (query dictations) from 17 participants over the span of 120 days. 15 of the 17 participants opted to complete our post-participation survey.

Recording time

Running a generalized linear model using recording time as dependent variable and query attributes (number of functions, number of tables, number of projections) as independent variables revealed that speakql recording time increased by .06 (6 percent) for every table in the function ($p = 0.005$), and decreased by .17 (17 percent) ($p = 0.001$) for every function in the query.

Number of Attempts

On average, SpeakQL queries required .43 (43 percent) more attempts than SQL queries ($p = 0.016$). However, the percent difference decreases by .24 (24 percent) for queries with at least one function ($p = 0.005$) and .07 (7 percent) per table referenced in the query ($p = .07$).

Survey feedback

After user study session completion, we asked participants to fill out a voluntary survey that asked them which features they used, and for the features that they used, how useful they found them. For features that they did not use, we asked for reasons why they did not use the feature. Figure 2.14 shows the current feedback that participants provided when asked to determine a features usefulness when compared to SQL. While most feedback was either neutral or positive, suggesting a general 'niceness' factor on the part of the participants where participants

	Frequency
– Positive –	**31**
General Positive Impressions	12
Positive Unbundling Impressions	14
Positive Natural Function Impressions	4
Positive Ordering Impressions	1
– Negative –	**23**
Syntax Difficulty	9
Unfamiliar Language, Needs Practice	8
Faux Natural Language	6
– Improvement Ideas –	**10**
Minimize Punctuation	5
Syntax Improvements	5

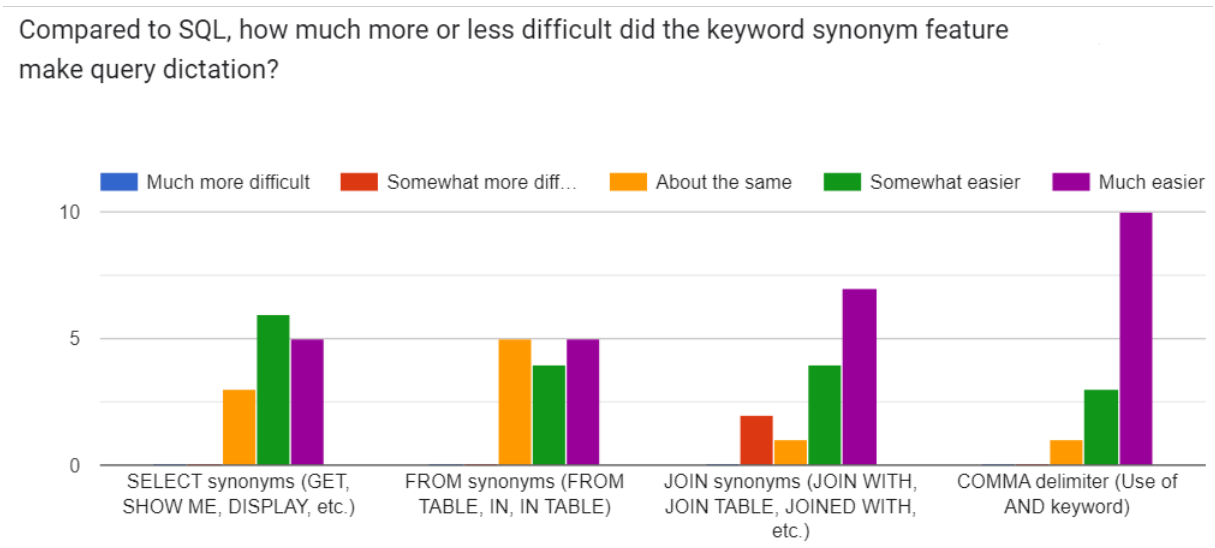
Table 2.2. Thematic Analysis - Category and Code Frequencies

may be hesitant to provide critical feedback believing that the research facilitator may have been involved with developing the SpeakQL dialect; some features received more positive responses than others, suggesting that those with more positive responses were actually perceived positively by the participants. These *relatively positive* features include natural functions, unbundling, comma and join synonyms, and modifier ordering for queries that required at least three modifier expressions.

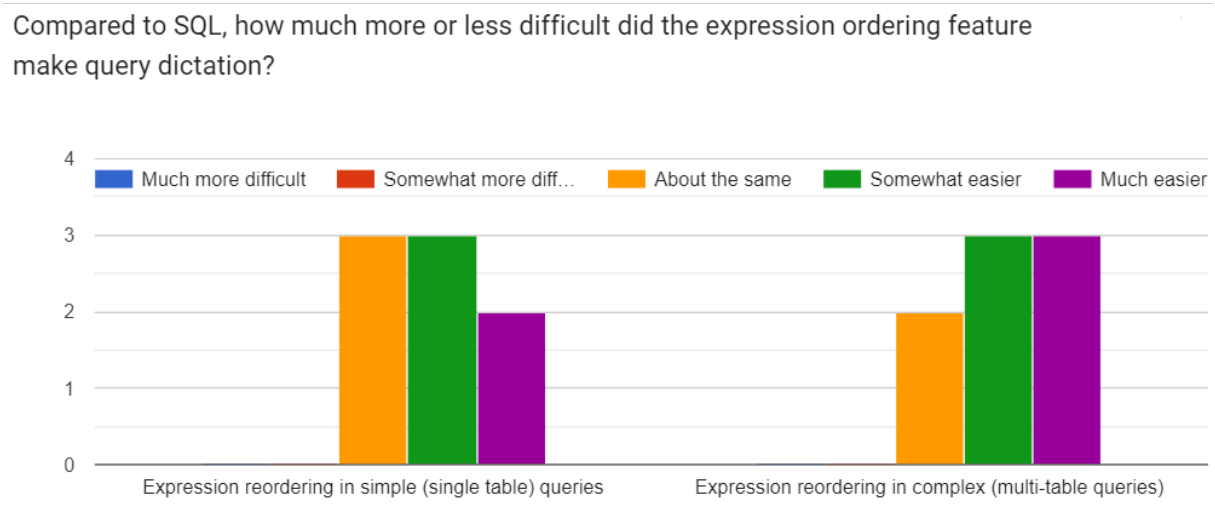
According to survey feedback, expression and modifier ordering are currently the least-used features. The majority of survey respondents who indicated that they did not use these features stated that their decision was due to their familiarity and comfort with SQL syntax. The other predominate reason participants opted out of using SpeakQL features was their confusion with the syntax or inability to remember the correct usage criteria.

2.7 Conclusion

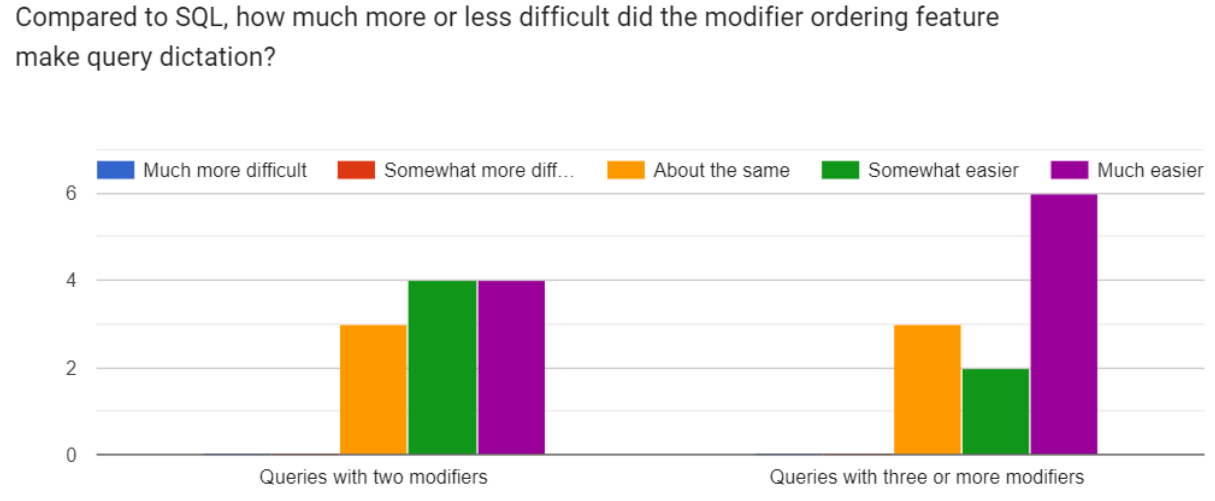
SpeakQL is an ongoing project with the objective of improving the experience of querying arbitrary relational databases using speech-based technology. We used the preliminary work on SpeakQL [13] as a starting point to develop the SpeakQL dialect, which is an extension of



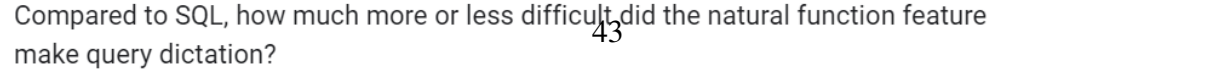
(a) Keyword Synonyms



(b) Expression Ordering



(c) Modifier Ordering



SQL formal grammar to make the language feel more natural for users while retaining SQL's formality and expressive power.

Work on SpeakQL is ongoing including implementation of automatic speech recognition features, application of SpeakQL 1.0 structural and literal determination methods to the more verbose SpeakQL dialect, and integration of user feedback from the ongoing user study. As we continue developing SpeakQL 2.0 toward these objectives, we intend to perform additional user studies to measure the effectiveness of SpeakQL's updates quantitatively and qualitatively based on feedback from its intended user populations.

Additional research areas related to SpeakQL and the SpeakQL language dialect include the study of mixed initiative user interfaces to enable stateful and incremental query building, error correction and disambiguation through human-computer dialogue, and parser and translation performance optimization to enable near-real-time interaction using more complex grammars such as the SpeakQL dialect.

Chapter 3

SNAILS

Chapter 4

SKALPEL

Chapter 5

Related Work

Chapter 6

Conclusion and Future Work

Appendix A

Heading on Level 0 (chapter)

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

A.1 Heading on Level 1 (section)

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

This is the second paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no

information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

And after the second paragraph follows the third paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

After this fourth paragraph, we start a new paragraph sequence. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content,

but the length of words should match the language.

A.1.1 Heading on Level 2 (subsection)

This is the second paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

And after the second paragraph follows the third paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

After this fourth paragraph, we start a new paragraph sequence. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Hello, here is some text without a meaning. This text should show what a printed text

will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

This is the second paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Heading on Level 3 (subsection)

And after the second paragraph follows the third paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

After this fourth paragraph, we start a new paragraph sequence. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like

this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

This is the second paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

And after the second paragraph follows the third paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Heading on Level 4 (paragraph)

After this fourth paragraph, we start a new paragraph sequence. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

This is the second paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

And after the second paragraph follows the third paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like

this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

After this fourth paragraph, we start a new paragraph sequence. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

A.2 Lists

A.2.1 Example for list (itemize)

- Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

- Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected

font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

- Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.
- Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.
- Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Example for list (4*itemize)

- Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.
  - Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.
    - Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.
      - Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no

information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

  - Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

    * Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

- Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information.

Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

- Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

A.2.2 Example for list (enumerate)

1. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

2. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected

font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

3. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

4. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

5. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Example for list (4*enumerate)

1. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

   (a) Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

      i. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

         A. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no

information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

B. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

ii. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

(b) Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information.

Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

2. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

A.2.3 Example for list (description)

First item in a list Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Second item in a list Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense

like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Third item in a list Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Fourth item in a list Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Fifth item in a list Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This

text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Example for list (4*description)

First item in a list Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

First item in a list Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

First item in a list Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain

all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

First item in a list Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Second item in a list Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Second item in a list Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how

the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Second item in a list Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Second item in a list Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Bibliography

[1] grammars-v4. https://github.com/antlr/grammars-v4, 2022.

[2] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[3] M. M. Astrahan and D. D. Chamberlin. Implementation of a structured english query language. *Commun. ACM*, 18(10):580–588, October 1975.

[4] D. D. Chamberlin, M. M. Astrahan, K. P. Eswaran, P. P. Griffiths, R. A. Lorie, J. W. Mehl, P. Reisner, and B. W. Wade. Sequel 2: A unified approach to data definition, manipulation, and control. *IBM J. Res. Dev.*, 20(6):560–575, November 1976.

[5] Donald D. Chamberlin and Raymond F. Boyce. Sequel: A structured english query language. In *Proceedings of the 1974 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control*, SIGFIDET '74, page 249–264, New York, NY, USA, 1974. Association for Computing Machinery.

[6] Brad Kelechava. The sql standard - iso/iec 9075:2016 (ansi x3.135), Jul 2020.

[7] Hyeonji Kim, Byeong Hoon So, Wook Shin Han, and Hongrae Lee. Natural language to sql: Where are we today? *Proceedings of the VLDB Endowment*, 13:1737–1750, 6 2020.

[8] Gabriel Lyons, Vinh Tran, Carsten Binnig, Ugur Cetintemel, and Tim Kraska. Making the case for query-by-voice with echoquery. volume 26-June-2016, pages 2129–2132. Association for Computing Machinery, 6 2016.

[9] I. Scott MacKenzie. *Human-Computer Interaction: An Empirical Research Perspective*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2013.

[10] Terence Parr, Sam Harwell, and Kathleen Fisher. Adaptive ll(*) parsing: The power of dynamic analysis. In *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA '14, page 579–598, New York, NY, USA, 2014. Association for Computing Machinery.

[11] P. Reisner. Use of psychological experimentation as an aid to development of a query language. *IEEE Transactions on Software Engineering*, SE-3(3):218–229, May 1977.

[12] Phyllis Reisner, Raymond F. Boyce, and Donald D. Chamberlin. Human factors evaluation of two data base query languages: Square and sequel. In *Proceedings of the May 19-22, 1975, National Computer Conference and Exposition*, AFIPS '75, page 447–452, New York, NY, USA, 1975. Association for Computing Machinery.

[13] Vraj Shah, Side Li, Arun Kumar, and Lawrence Saul. Speakql: Towards speech-driven multimodal querying of structured data. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 2363–2374, 2020. School: UCSD.

[14] Vraj Shah, Side Li, Arun Kumar, and Lawrence Saul. Speakql: Towards speech-driven multimodal querying of structured data, 2020.

[15] J. N. Towse and G. J. Hitch. Is there a relationship between task demand and storage space in tests of working memory capacity? *The Quarterly Journal of Experimental Psychology Section A*, 48:108–124, 2 1995.

[16] Andre Vandierendonck Gino De Vooght. Working memory constraints on linear reasoning with spatial and temporal contents. *The Quarterly Journal of Experimental Psychology Section A*, 50(4):803–820, 1997. PMID: 9450380.