Semantic Search in Relational Databases

A Major Qualifying Project submitted to
the faculty of Worcester Polytechnic Institute
in partial fulfillment of the requirements for the
Degree of Bachelor of Science in Computer Science

Submitted by:
Thomas Finelli and Yo Karita

Professor Mohamed Y. Eltabakh, Project Advisor

December 15, 2016

1 Abstract

Most relational databases are accessed and managed by a form of Structured Query Language (SQL). For new users, learning SQL takes time and can be complex. This project investigates using natural language as the initial input for searching a relational database. The natural language query is converted into one or more valid SQL queries which can be executed on a database. This system takes advantage of an ontology and an ontology-to-database mapping. Ontologies allow relationships between natural language, resulting in a deeper understanding. The ontology-to-database mapping aids in connecting ontology keywords directly with parts of the database schema.

Table of Contents

2 Introduction

Relational databases are commonly used to store all different types of data. In order to interact with a relational database, some defined, structured language must be used. One of the most commonly used languages for database management is Structured Query Language (SQL).

If a user wants to input, retrieve, or change the data in SQL based relational database, then they must use the Structured Query Language. Learning and understanding SQL takes time and it can be complex. Interacting with a database would be much simpler if the user could type in what they wanted using keywords from a natural language rather than SQL.

There are, however, some challenges that arise when using a natural language. One is that natural languages can be ambiguous. Another is that the user may not know the exact schema of the database, and as a result, they may use other similar terms in their natural language query (NLQ). It can be challenging to make the connection between the user's similar terms and the particular terms used in the database schema.

The literature discusses some approaches that can be used to help bridge the user's keywords and the database schema's keywords (Saha, et al., 2016). One is by utilizing an ontology and an ontology-to-database mapping. An ontology can be defined as: "a model for describing the world that consists of a set of types, properties, and relationship types" (Garshol, 2004). In other words, ontologies can be used to connect objects together through relationships. An ontology-to-database mapping relates keywords in the ontology directly to the database schema. These mappings usually refer to a particular database table name or a table's column name. Overall, these two concepts can be used to better understand the user's NLQ.

2.1 Project Goal

The main goal of this project is to design a system that is capable of transforming a user's NLQ into one or more valid SQL queries by using a semantic based approach which utilizes an ontology and ontology-to-database mapping. As mentioned, the system will a list of SQL queries for a single user NLQ. Each of these returned SQL queries will have a respective probability value. This probability value indicates a theoretical correctness, that is, how likely the system generated SQL query coincides with the user's NLQ. From here the user would be able to manually select the SQL query that is closest to what they intended from the list of generated SQL queries. The selected SQL query would then be executed on the database, and the query results are displayed in table format.

In order to test our system we needed to create a database, and then populate it with tables and data. We arbitrarily chose to create a database containing travel data, however, a database containing nearly any type of data would also work. There are four types of travel data that are the main focuses of the database: flights, hotels, tours, and car rentals. The database also contains customer data which is linked to one of the travel data type by a reservation (e.g. a customer reserves a hotel room or books a flights).

3 Background

The two most important components of the NLQ to SQL system are the ontology and the ontology to database mapping. The ontology sets the scope on which keywords are searchable and the mapping connections these keywords directly to the database.

3.1 Ontology

An ontology can be defined as a structural way of relating objects or classes and their properties to other objects or classes and their properties. As described in our definition of an ontology, there are four ontology components that we focus on: objects, classes, properties, and relationships.

- **Objects** represents things, which can be physical or abstract; these are the basic building blocks of an ontology
- **Classes** are a generic concept which Objects may belong to
- **Properties** are attributes that an Object or Class may have
- **Relationships** connect one object to another through some commonality

It is important to note that these particular words used to describe an ontology can vary from one person's definition to another. However, the overall principals remain fairly constant between definitions.

Ontologies typically have an overlying topic which sets the scope of the ontology. An example of a simple animal ontology can be found in Figure 1. In the figure, the classes are: animal, carnivore, and herbivore; the objects are: lion, antelope, and plant; the properties are: gender; and the relationships are: eats and is a. Note that the distinction between an object and a class can vary depending on context. For example if the example ontology was more detailed, then "Congo lion" could be added as an object of "Lion", which transitions "Lion" into a class.

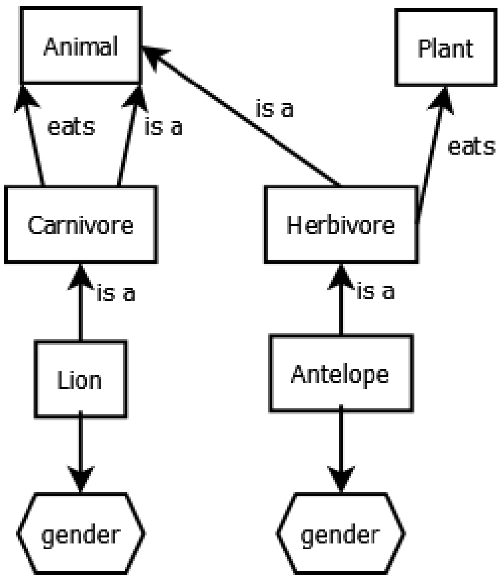


Figure 1: A simple example ontology about animals. Rectangular boxes represent classes and objects. Hexagons represent attributes. Labeled arrows represent relationships. This diagram was adapted from a diagram by (White, 2005)

There are many different ways to implement an ontology. One of the most common standards is the Web Ontology Language (OWL) (Bechhofer, et al., 2014). This standard has its own particular definitions for an ontology class, property, and object; which are similar to what we previously described. It also defines a language that can be used to generate relationships which then result in an ontology. For our system, we adopt a simplified version of the OWL standards. Specifically, we currently only utilize objects (also known as individuals in OWL), classes, and the "is a" relationship.

3.2 Ontology to Database Mapping

An ontology to database mapping can be used to connect words in the ontology to particular part of the database schema. The mapping essentially allows the ontology to be independent of the database schema (Saha, et al., 2016). This is useful because it means that this approach can work with existing databases, and no major modifications need to be made to the schema. It also allows for the ontology to be as simple or detailed as desired. For example, in a detailed ontology, multiple different ontology keywords (which may be synonymous) can map to the same part of the schema. Figure 2 illustrates the concept of mapping. Essentially, the ontology keywords can map to specific tables or columns in the database schema.

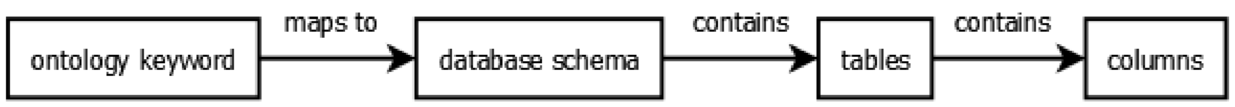


Figure 2: The ontology to database mapping process.

3.3 Summary

An ontology can be quite useful when analyzing natural language. Its use of relationships enables traversal from one concept or object to another, and thus it can provide a deeper understanding of natural language. Additionally, the ontology to database mapping is crucial for determining how the NLQ relates to the database schema.

4 Methodology: Setting Up the Database, Ontology, and Database-to-Ontology Mapping

An overview of the entire system's process is illustrated in Figure 3. This part of the methodology describes the setup for the database, ontology, and ontology-to-database mapping (i.e. the tools shown on the far right side of Figure 3). The system steps numbered 1 through 7 on Figure 3 will be discussed later on in a subsequent methodology section.

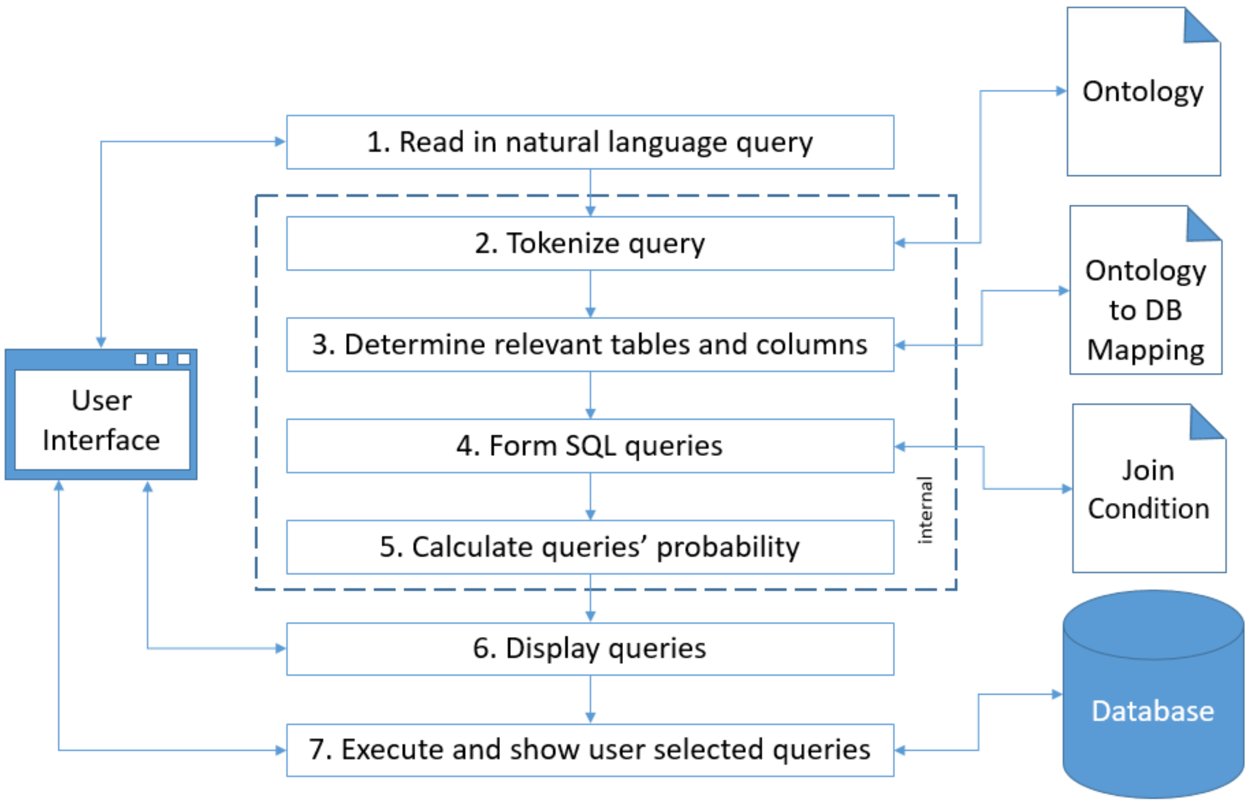

Ontology
1. Read in natural language query
2. Tokenize query
Ontology to DB Mapping
3. Determine relevant tables and columns
User Interface
4. Form SQL queries
Join Condition
internal
5. Calculate queries' probability
6. Display queries
Database
7. Execute and show user selected queries

Figure 3: Process flow diagram for the system. The numbered steps (1-7) demonstrate how a user query is transformed into a SQL query. The system interacts with the external database and files (far right).

4.1 Database Overview

Before running the semantic search process, a database must be created and populated with data. Nearly any kind of data would work, but for this project we chose to use travel data. There are four types of travel data that are the main focuses of the database: flights, hotels, tours, and car rentals. The database also contains customer data which is linked to one of the travel data type by a reservation (e.g. a customer reserves a hotel room or books a flights). The database schema is shown in Figure 4.

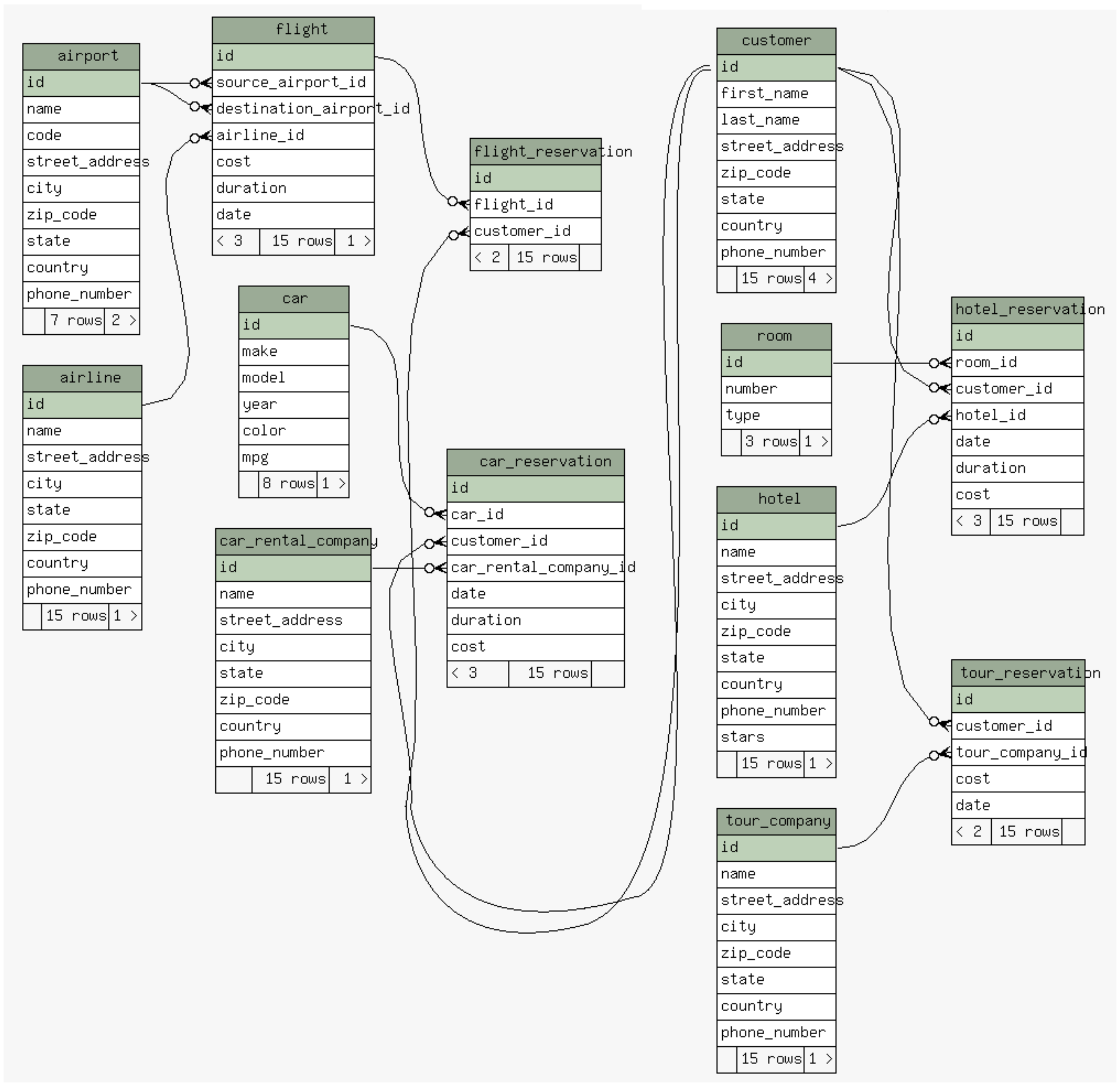


Figure 4: The travel database schema. Arrows represent foreign key relationships.

The database was designed to be viewed from a historical prospective. More specifically, when a user searches the database, they are looking at trips, vacations, and reservations that were made in the past by other customers or users.

PostgreSQL was chosen as the database management system. The primary reason for using PostgreSQL is due to its free and open source codebase, which allows for modifications if desired. It is also widely used and has sufficient documentation. The official PostgreSQL website mentions a study which showed that 30% of tech companies used PostgreSQL for their core applications in 2012 (PostgreSQL: PostgreSQL Press FAQ, n.d.).

4.2 Database Tables

There are four main tables related to travel: flight, hotel, car rental, and tour. Additionally, there is one table that contains customer information. For example it has customers' names and addresses. Each of the 4 travel type tables has corresponding reservation table (i.e. flight_reservation, hotel_reservation, car_reservation, and tour_reservation). Each of these reservation tables have two foreign key relationships. One of the foreign keys connects to the reservations tables' respective travel type (e.g. flight, hotel, car rental, and tour), and the other foreign key connects the reservation to a customer ID in the customer table. A summary of the database tables can be found in Table 1, Table 2, Table 3, Table 4, and Table 5.

Table 1: Descriptions of tables related to flights

Database Table	Description
flight	Flight information such as the destination *airport*, the *airline*, and cost of the flight, and the date on which the flight departs
flight_reservation	Connects a particular *flight* with a *customer* through its foreign keys
airport	Information about the airport such as the name and location (city, country, etc.)
airline	Information about the airline such as the name and location (city, country, etc.)

Table 2: Descriptions of tables related to cars

Database Table	Description
car	Contains data about different types of cars such as make, model, color, etc.
car_reservation	Connects a particular *car* and *car_rental_company* with a *customer* through its foreign keys

car_rental_company	Information about the car rental company such as its name and location

Table 3: Descriptions of tables related to hotels

Database Table	Description
hotel	Information about the hotel such as its name, location, and rating
hotel_reservation	Connects a particular *hotel* and room with a *customer* through its foreign keys
room	Contains information about a hotel room such as the type of room and room number

Table 4: Descriptions of tables related to tours

Database Table	Description
tour_reservation	Connects a particular *tour_company* with a *customer* through its foreign keys
tour_company	Information about the tour company such as its name and location (city, country, etc.)

Table 5: Descriptions of tables related to customers

Database Table	Description
customer	Contains customer information such as customers' name and address

4.3 Database Views

When performing a search it is desirable to connect the tables that have foreign key relationships for the majority of cases. For example, when searching for "flight to boston" the user most likely wants to see lots of information such the date of the flight, the origin and destination airports, etc. However, this information is distributed in multiple different tables. For example, the date of the flight is located in the *flight* table, but the airport information is located in the *airport* table.

Most searches will require aggregating data from multiple tables. Since this will be a common occurrence, we utilize database views. We created 5 different database views, each of which represents a SQL query that joins similar tables. The views and their column names are presented in Figure 5 and a summary of each view and description of the tables joined are presented in Table 6.

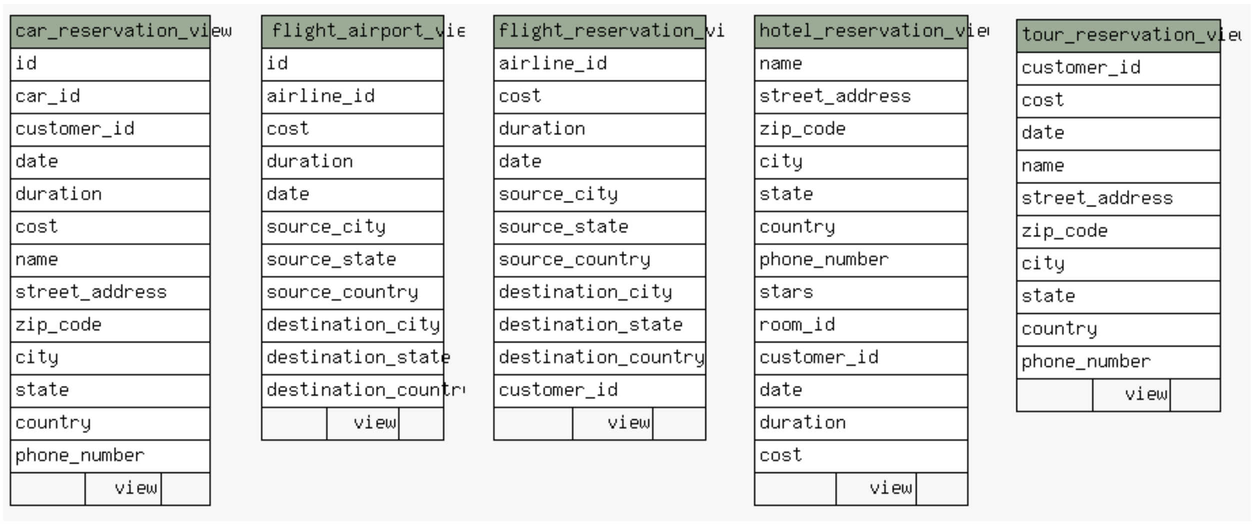


Figure 5: Schema for database views

Table 6: Descriptions of database views

View Name	Description
car_reservation_view	Joins *car* and *car_rental_company* through *car_reservation*
flight_airport_view	Joins *flight* and *airport* using both the airport source and destination
flight_reservation_view	Joins *flight_airport_view* and *flight_reservation*
hotel_reservation_view	Joins *hotel* and *room* through *hotel_reservation*
tour_reservation_view	Joins *tour_reservation* and *tour_company*

4.4 Population of the Database

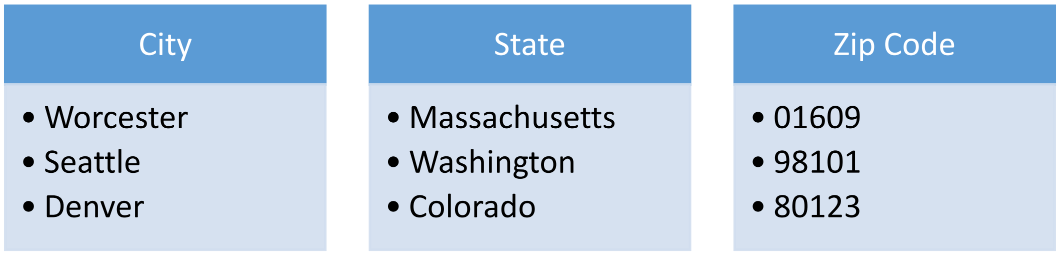
Prior to performing a semantic search, the database needs to be populated with data. The goal was to do this in a quick manner and with fairly realistic data. Lists of popular or common data were created for many of the table columns. For example, a list the most common street names in the U.S. was created for use in addresses. Then one entry from the list is selected and used to generate a complete address. The address is used, in combination with other data, to create a single row which is inserted into one of the database tables. The generic process is as follows:

1. Gather lists of common data (e.g. street names, car brands, airports, etc.)
2. Randomly select an entry from one of the lists
3. Use the entry to generate a complete set of data which can then be inserted into one of the database's tables

This process is quick and automatic. It eliminates the time and effort cost of manually generating data. The majority of the data that is used in the database was generated in this fashion.

4.4.1 Grouped and Related Data

Some data only makes sense when grouped together. For example location data such as city, state, and zip code need to be generate as a group. To illustrate this concept, let us examine 3 lists of data: cities, states, and zip codes (Figure 6). It makes sense that Worcester, Massachusetts 01609 goes together as it is a real, valid location. In contrast, if we were to randomly select data from each list we could come up with something like Worcester, Colorado 98101 which does not make sense as there is no city named Worcester in the state of Colorado, and the specified zip code does not match the correct city. For more examples see Table 7.



City	State	Zip Code
• Worcester • Seattle • Denver	• Massachusetts • Washington • Colorado	• 01609 • 98101 • 80123

Figure 6: Example lists of cities, states, and zip codes.

Table 7: Examples of combined data, where each data piece was randomly chosen from one of the lists.

Location	Valid	Description
Worcester, Massachusetts 01609	O	Real, valid location
Worcester, Colorado 98101	X	There is no city named Worcester in the state of Colorado, and the specified zip code does not match the correct city
Seattle, Washington 01609	X	The city and state match, but the zip code belongs to a different city

As a solution, some of the data is generated in a grouped manner. Individual classes were created in the Java code, such as Location and Airport. These classes can return a grouping of data which can be used alongside other data to generate a realistic entry in a database table.

Note that street names and street numbers can also be considered grouped data. However, the smallest, most precise piece of location data that our ontology supports is the city. Therefore we ignore grouped data for anything smaller (e.g. street name, street number, zip code), and we do use grouped location data for location scales that are equal to a city and above (e.g. city, state, country). This scope is defined by the ontology and can be changed if the ontology is also changed.

4.5 Join Condition Overview

In order to combine data from multiple database tables some join condition must be specified in the resulting SQL query. We have created a spreadsheet which contains all of the join conditions for each of the tables. We currently only use one join condition for each of the table pairings, but this can be changed if desired. A screen capture of the join condition spreadsheet can be found in Figure 7.

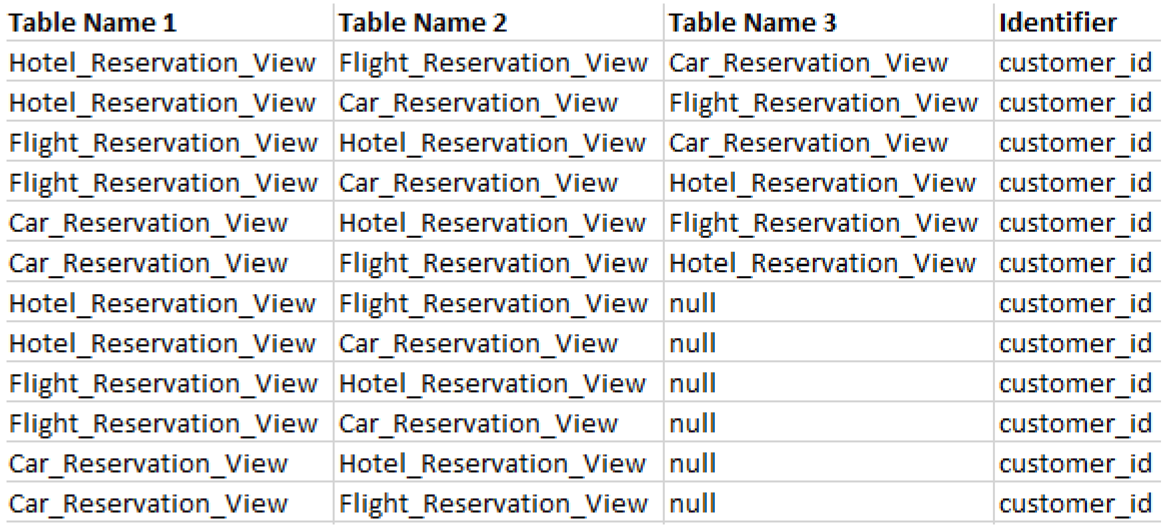


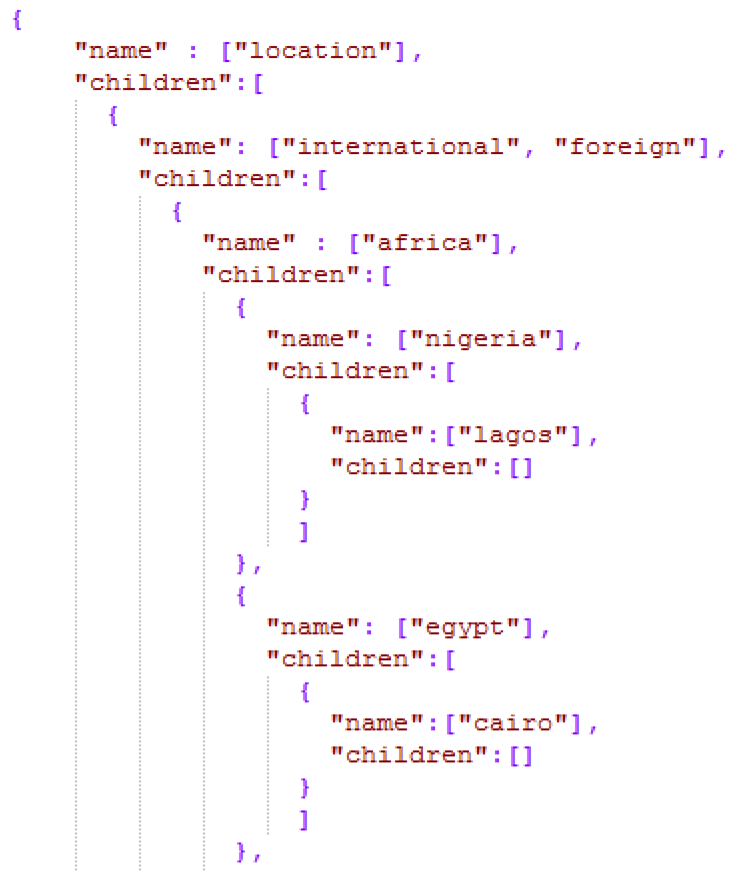
Table Name 1	Table Name 2	Table Name 3	Identifier
Hotel_Reservation_View	Flight_Reservation_View	Car_Reservation_View	customer_id
Hotel_Reservation_View	Car_Reservation_View	Flight_Reservation_View	customer_id
Flight_Reservation_View	Hotel_Reservation_View	Car_Reservation_View	customer_id
Flight_Reservation_View	Car_Reservation_View	Hotel_Reservation_View	customer_id
Car_Reservation_View	Hotel_Reservation_View	Flight_Reservation_View	customer_id
Car_Reservation_View	Flight_Reservation_View	Hotel_Reservation_View	customer_id
Hotel_Reservation_View	Flight_Reservation_View	null	customer_id
Hotel_Reservation_View	Car_Reservation_View	null	customer_id
Flight_Reservation_View	Hotel_Reservation_View	null	customer_id
Flight_Reservation_View	Car_Reservation_View	null	customer_id
Car_Reservation_View	Hotel_Reservation_View	null	customer_id
Car_Reservation_View	Flight_Reservation_View	null	customer_id

Figure 7: The join condition table. The field for the join condition is specified in the "Identifier" column. This specifies what the join condition should be when combining different tables.

4.6 Ontology Overview

In order for our system to function the database needs to be supplemented with an ontology. Therefore, an ontology was created with keywords that can be associated with travel data in the database. The ontology also sets the scope for valid keywords. In other words, only keywords that can be found in the ontology are searchable in our system. Any other words are discarded from the process.

The ontology was created in JSON file format. The system reads in the ontology file and creates a tree like structure in memory, making the ontology searchable. The ontology contains the following major categories: generic trip keywords (e.g. vacation, hotel, flight, etc.); locations (e.g. city, country, etc.); time (e.g. months, seasons, etc.); cost (e.g. cheap); and operators (e.g. less than, more than, etc.). An excerpt of ontology location data can be found in Figure 8.



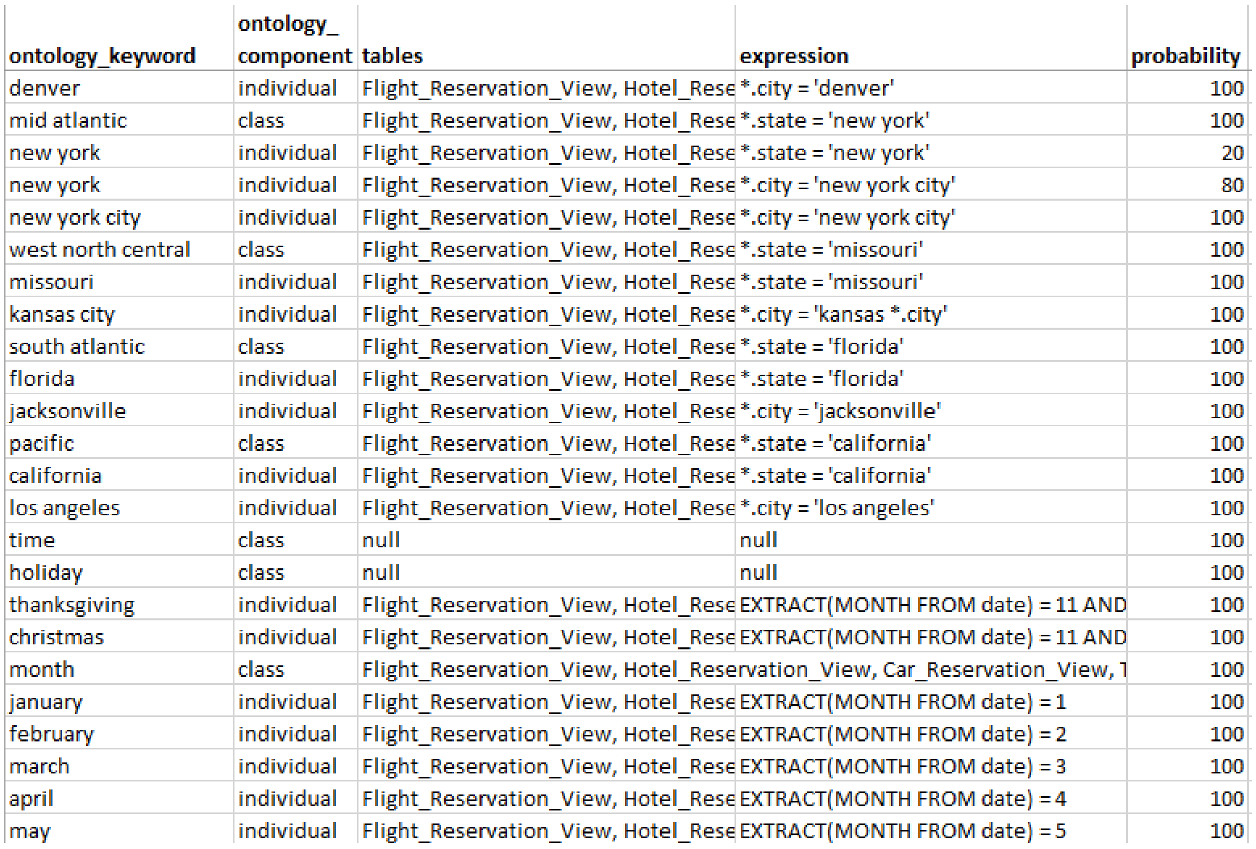
```
{
    "name" : ["location"],
    "children":[
      {
        "name": ["international", "foreign"],
        "children":[
          {
            "name" : ["africa"],
            "children":[
              {
                "name": ["nigeria"],
                "children":[
                  {
                    "name":["lagos"],
                    "children":[]
                  }
                  ]
              },
              {
                "name": ["egypt"],
                "children":[
                  {
                    "name":["cairo"],
                    "children":[]
                  }
                  ]
              },
```

Figure 8: Excerpt of the ontology file which is encoded in JSON format. This excerpt specifically shows location keywords and their respective relationships. The rest of the ontology follows this format.

The ontology supports parent-child relationships, as demonstrated by the "children" fields in Figure 8. Some keywords may be synonymous; these words are kept together in a comma separated array (see ["international", "foreign"] in Figure 8).

4.7 Ontology to Database Mapping

A mapping from the ontology to the database was created in order to connect the two together. For every keyword in the ontology, we created a mapping entry which then refers to database tables and tables' columns. The ontology to database mapping exists as a table with five columns: ontology keyword, ontology component, tables, expression, and probability. Each of these columns is described in detail in the following sections. An excerpt of the mapping can be seen in Figure 9.



ontology_keyword	ontology_ component	tables	expression	probability
denver	individual	Flight_Reservation_View, Hotel_Rese	*.city = 'denver'	100
mid atlantic	class	Flight_Reservation_View, Hotel_Rese	*.state = 'new york'	100
new york	individual	Flight_Reservation_View, Hotel_Rese	*.state = 'new york'	20
new york	individual	Flight_Reservation_View, Hotel_Rese	*.city = 'new york city'	80
new york city	individual	Flight_Reservation_View, Hotel_Rese	*.city = 'new york city'	100
west north central	class	Flight_Reservation_View, Hotel_Rese	*.state = 'missouri'	100
missouri	individual	Flight_Reservation_View, Hotel_Rese	*.state = 'missouri'	100
kansas city	individual	Flight_Reservation_View, Hotel_Rese	*.city = 'kansas *.city'	100
south atlantic	class	Flight_Reservation_View, Hotel_Rese	*.state = 'florida'	100
florida	individual	Flight_Reservation_View, Hotel_Rese	*.state = 'florida'	100
jacksonville	individual	Flight_Reservation_View, Hotel_Rese	*.city = 'jacksonville'	100
pacific	class	Flight_Reservation_View, Hotel_Rese	*.state = 'california'	100
california	individual	Flight_Reservation_View, Hotel_Rese	*.state = 'california'	100
los angeles	individual	Flight_Reservation_View, Hotel_Rese	*.city = 'los angeles'	100
time	class	null	null	100
holiday	class	null	null	100
thanksgiving	individual	Flight_Reservation_View, Hotel_Rese	EXTRACT(MONTH FROM date) = 11 AND	100
christmas	individual	Flight_Reservation_View, Hotel_Rese	EXTRACT(MONTH FROM date) = 11 AND	100
month	class	Flight_Reservation_View, Hotel_Reservation_View, Car_Reservation_View, T		100
january	individual	Flight_Reservation_View, Hotel_Rese	EXTRACT(MONTH FROM date) = 1	100
february	individual	Flight_Reservation_View, Hotel_Rese	EXTRACT(MONTH FROM date) = 2	100
march	individual	Flight_Reservation_View, Hotel_Rese	EXTRACT(MONTH FROM date) = 3	100
april	individual	Flight_Reservation_View, Hotel_Rese	EXTRACT(MONTH FROM date) = 4	100
may	individual	Flight_Reservation_View, Hotel_Rese	EXTRACT(MONTH FROM date) = 5	100

Figure 9: Excerpt of the ontology to database mapping. This mapping exists in a tab separated spreadsheet (.tsv) format. Each ontology keyword has an ontology component, tables, expression, and probability.

Some examples of mappings are provided at the end of this section.

4.7.1 Ontology Keyword

First, each keyword in the ontology is put into the mapping. When performing a lookup on the mapping the keyword is what is typically searched for, and the other columns in the mapping are returned as a result.

4.7.2 Ontology Component

This component is based on the Ontology Web Language (OWL) specification types. There are two types that are utilized: class and individual. We define class as a node that has children, and individual that has no children. The ontology component column is not currently utilized in the SQL generation process. However, it may be useful in future work.

4.7.3 Tables

This column consists of all the possible database tables or views that the ontology word may reference. The entry for tables may be listed as null if there is no direct connection between the ontology keyword and a database table. This is frequently null for class ontology components as they tend to be generic. If the table field is null, then the expression also must be null. Additionally, if the table field is null, then this is equivalent to the ontology keyword not existing in the mapping, thus it technically could be omitted from the mapping completely. We have opted to include all the ontology keywords in the mapping, even if they will have null values for table, to be complete.

4.7.4 Expression

The expression is the direct connection between the ontology keyword and the database column. It is the same as the WHERE clause in a standard SQL query. This may be null if the ontology keyword does not directly reference a column in one of the database tables. The expression is frequently null for class ontology components as they tend to be generic.

4.7.5 Probability

The probability value is the base likeliness for a particular ontology keyword's expression. For example, the ontology keyword "New York" can refer to the state or the city, so different base probabilities can be assigned to each of these meanings separately. Currently, probabilities are set to 100% if there is only one expression or meaning for a keyword. Otherwise, for multiple expressions (and thus multiple mapping entries), the probabilities must sum to 100%. The actual value for individual mappings when there are multiple expressions can be set to any desired value, as long as they sum to 100%.

4.7.6 Mapping Examples

Some examples are provided in Table 8 to illustrate the process of creating a mapping. The first example in the table is the ontology keyword "Denver". Denver refers to a location, specifically the city located in the state of Colorado. All of the major views (flight, hotel, car, tour) have locations, and more specifically they each have a column named "city" in them, so we can list all of these views as valid tables. Next is the expression which lists the WHERE clause. Again, all of the views have a "city" column, so the expression can refer to this directly. In the next example "hotel" is the ontology keyword. In this case, the only view that makes sense is the "Hotel_Reservation_View", because no other views contain data about hotels. Lastly the expression has to be null, since hotel is a generic ontology keyword and does not directly refer to any of the view/table columns.

Table 8: Examples of ontology to database mappings.

Ontology keyword	Tables	Expression
Denver	Flight_Reservation_View, Hotel_Reservation_View, Car_Reservation_View, Tour_Reservation_View	*.city = 'denver'
hotel	Hotel_Reservation_View	null

5 Methodology: Transforming a Natural Language Query into Valid SQL Queries

An overview of the entire system's process is illustrated again in Figure 10. This part of the methodology describes the process for transforming a user's natural language query into multiple valid SQL queries (i.e. the steps listed as numbers 1 through 7 in Figure 10).

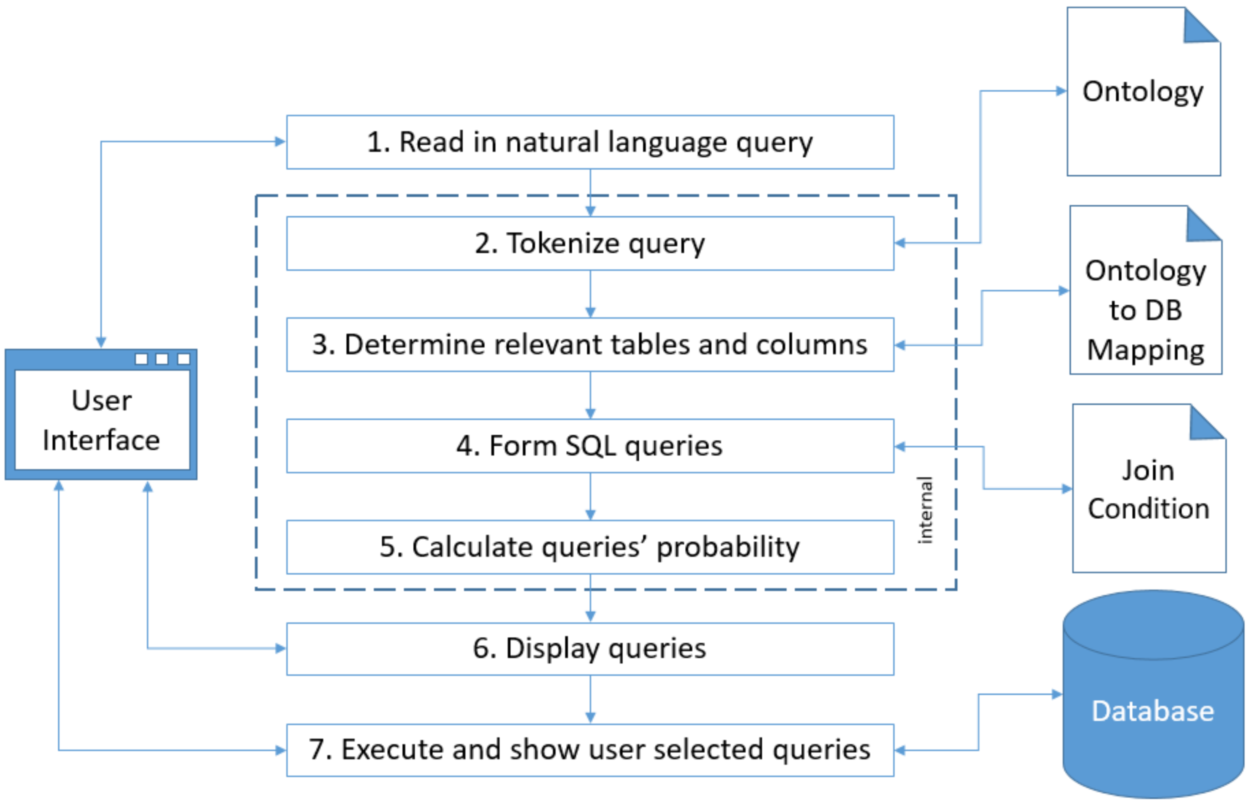

1. Read in natural language query
2. Tokenize query
3. Determine relevant tables and columns
4. Form SQL queries
5. Calculate queries' probability
6. Display queries
7. Execute and show user selected queries
User Interface
internal
Ontology
Ontology to DB Mapping
Join Condition
Database

Figure 10: Process flow diagram for the system. The numbered steps (1-7) demonstrate how a user query is transformed into a SQL query. The system interacts with the external database and files (far right).

5.1 Parsing and Tokenizing the User's Natural Language Query

The first step is to read in and parse the user's natural language query. Initially the NLQ is tokenized based on spaces. Next each word is looked up in the ontology. If a word cannot be found in the ontology, then the system will try again using the original word plus the following word (i.e. for phrases with two words). At this point any words in the NLQ that cannot be found in the ontology are discarded. An illustration of this process can be found in Figure 11 using the NLQ “hotel in Denver during October”.

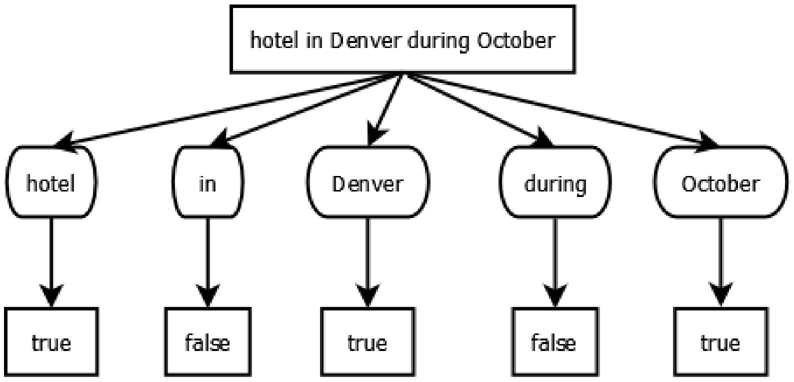


Figure 11: Example of tokenizing a user query, and then performing a search in the ontology for each token. If the token is found in the ontology, then the result is shown as “true”. Otherwise, the result is “false”. Words that are not found in the ontology are discarded.

The next step is to look up the valid keywords (i.e. those that could be found in the ontology) in the ontology to database mapping.

5.2 Returning the Mappings for a Tokenized Query

The mapping lookup process is performed on each of the tokenized ontology phrases. Recall that the tokenized ontology phrase is typically a single word (e.g. hotel, flight, Denver, Asia) but may be two words (e.g. New York). Each tokenized phrase will have a mapping returned, resulting in a list of mappings. This list of mappings is used in determining which tables and expressions need to be included in the generated SQL query.

5.3 Determining the Tables

In order to generate a SQL query we must first determine which database tables the user needs to access to fulfil their query. In essence this phase is about determining the FROM clause in a SQL query. To do this we examine the list of mappings that is generated from the tokenized ontology phrases. From the list of mappings we calculate the total frequency for each of tables that appears in the mappings.

From this frequency we can attempt to determine which tables the user will want data from. It is possible that the highest frequency table is not always what the user wants.

Therefore, we will provide multiple SQL queries which utilize different combinations of tables. This gives the user multiple options to choose from.

5.3.1 Calculating the Frequency and Probability for the Tables

We use the table frequency as an indication of table probability, that is, how likely it is that the user wants data from a particular table. For example, if the user inputs the query "hotel in Denver" the following mapping for tables is returned: hotel → Hotel_Reservation_View; in → null; Denver → Hotel_Reservation_View, Flight_Reservation_View, Car_Reservation_View, Tour_Reservation_View. Since Hotel_Reservation_View appears twice, it has a frequency value of 2; the other 3 views have a frequency value of 1. This frequency calculation is illustrated in Table 9.

Table 9: An example frequency calculation for the user query "hotel in Denver". Each token is looked up in the ontology to database mapping, and the resulting set of tables is tallied. Whichever view(s) is/are have the highest frequency are defined as base set (i.e. the set of tables/views that the user is most likely interested in).

Table Mapping	Mapping keyword frequency			Sum
	hotel	**in**	**Denver**	
Hotel_Reservation_View	1	0	1	2
Flight_Reservation_View	0	0	1	1
Car_Reservation_View	0	0	1	1
Tour_Reservation_View	0	0	1	1

Based on the frequency calculation we see that the user probably wants data from the hotel table, so this will be the highest ranked result. As previously mentioned, in most situations we return multiple SQL queries in case the highest ranked result is not what the user wants. To solve this we combine the highest ranked table(s) with the other tables and return each of these combinations as a possible SQL query interpretation. These resulting combinations are considered secondary results, and will be ranked with lower probability values when displayed to the user.

The combination process is illustrated in Figure 12 using the previously mentioned example query.

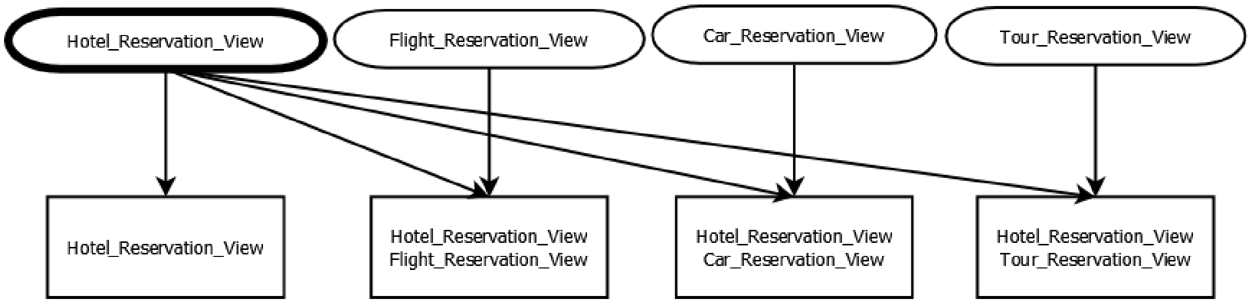

Hotel_Reservation_View
Flight_Reservation_View
Car_Reservation_View
Tour_Reservation_View
Hotel_Reservation_View
Hotel_Reservation_View Flight_Reservation_View
Hotel_Reservation_View Car_Reservation_View
Hotel_Reservation_View Tour_Reservation_View

Figure 12: An example of the combination process. The base table(s) (bold outline) is combined with each of the remaining tables to form four sets of table combinations.

Each of these combinations results in a unique set of tables, which in turn results in a unique SQL FROM clause. These FROM clauses will be combined with an expression (i.e. the WHERE clause), which is discussed in a subsequent section, to form a full SQL query.

5.4 Determining the Expressions

Each of the mappings generated from the tokenized ontology phrases may also have an expression associated with it. How the expressions are handled depends whether the user needs data from a single table or for multiple tables.

5.4.1 Single Table

Handling the expressions when only using a single table is fairly simple. The expressions are just combined together with SQL "AND". Take the example user query as "hotel in Denver during October" which results in the following expression mapping: hotel →null; in → null; Denver →*.city = 'Denver'; October → EXTRACT(MONTH FROM *.date) = 10.

In this case there are two expressions, which are combined to form *.city = 'Denver' AND EXTRACT(MONTH FROM *.date) = 10. If there was only one expression, there would be no AND necessary; and if there are more expressions, then additional ANDs would be appended along with each additional expression. Lastly the * in the resulting expression is replaced with the table name that is being queried. Afterwards, the expression can be directly substituted into the WHERE clause in a SQL query. This example is illustrated in Figure 13.

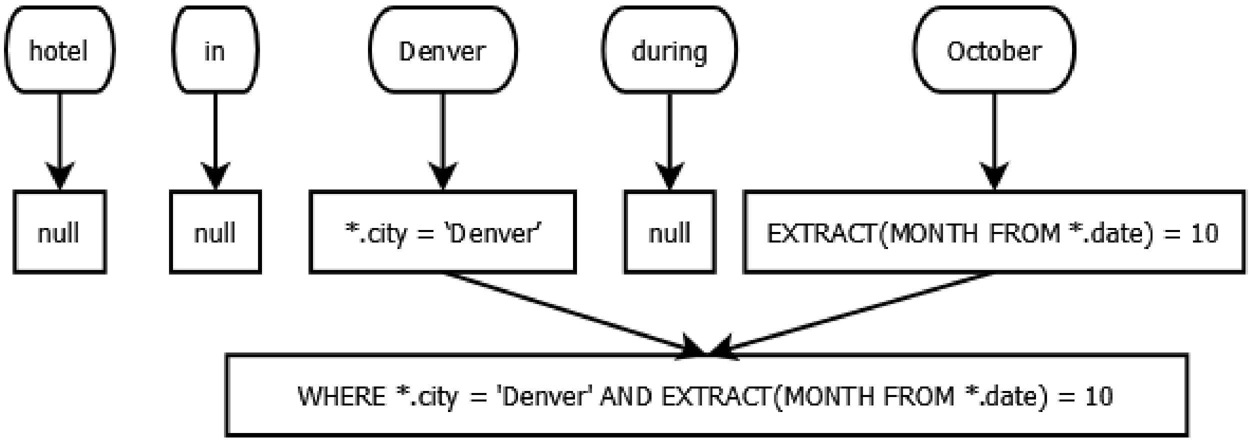

hotel
in
Denver
during
October
null
null
*.city = 'Denver'
null
EXTRACT(MONTH FROM *.date) = 10
WHERE *.city = 'Denver' AND EXTRACT(MONTH FROM *.date) = 10

Figure 13: Expression look up process for the example user query "hotel in Denver during October". Keywords that do not have an expression result in "null". Each individual expression is combined with SQL "AND" in order to form a final combined expression.

5.4.2 Multiple Tables

When dealing with multiple tables, the WHERE clause in a SQL query must contain at least one expression that acts as a join condition. We specify the join conditions for tables in a separate file which is in a tab separated spreadsheet (.tsv) format. Currently, the join condition is the same for all table combinations, but this can be modified if desired. For our setup we use the following combined expression as a join condition (from the join condition spreadsheet file):

$$WHERE\ \ Table\_base.customer\_id = Table\_i.customer\_id$$
$$AND\ \ Table\_base.date\ = Table\_i.date$$

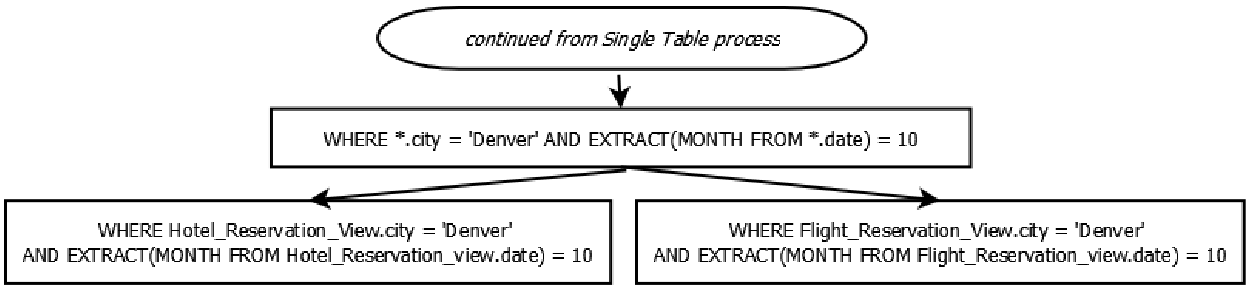
Where *Table_base* is arbitrarily chosen to be one of the tables and *Table_i* is some other table that is not the base table. This join condition is repeated for any additional tables (i.e. beyond two tables), where *Table_i* is replaced with each of the non-base tables. For example, if there were three tables involved, then the join condition would look as follows:

$$WHERE\ \ Table\_base.customer\_id = Table\_i.customer\_id$$
$$AND\ \ Table\_base.date\ = Table\_i.date$$

$$AND\ \ Table\_base.customer\_id = Table\_j.customer\_id$$
$$AND\ \ Table\_base.date\ \ = Table\_j.date$$

This example condition uses the three tables: *Table_base*, *Table_i*, and *Table_j*. If there were four tables, the condition would be append with *Table_k* and so on.

At this point the join condition is determined. However, we still need to examine the expressions returned for each of the mappings. The procedure for the mapping expressions with multiple tables is almost the same as the *Single Table* expression process. Refer to the *Single Table* expression section for details. The only difference is that the *Single Table* expression process needs to be repeated for each of the involved tables. In particular, the final step in the *Single Table* expression process is to replace the * in the mapping expression with the table name; this replacement operation needs to be done for each of the involved tables when dealing with multiple table. An example for the * replacement process is shown in Figure 14. This figure uses the same example query provided in the *Single Table* expression process. This example uses two tables: Hotel_Reservation_View and Flight_Reservation_View.



continued from Single Table process
WHERE *.city = 'Denver' AND EXTRACT(MONTH FROM *.date) = 10
WHERE Hotel_Reservation_View.city = 'Denver' AND EXTRACT(MONTH FROM Hotel_Reservation_view.date) = 10
WHERE Flight_Reservation_View.city = 'Denver' AND EXTRACT(MONTH FROM Flight_Reservation_view.date) = 10

*Figure 14: An example of the * replacement process when multiple tables are involved. In this example there are two tables: Hotel_Reservation_View and Flight_Reservation_View. The * in the original expression is replaced with each of these table names.*

Keywords with Multiple Expressions

There are some keywords which may have multiple mappings. Each of these mappings can map to a different expression. For example the keyword "New York" maps to the expression *.city = 'New York' and to the expression *.state = 'New York'.

In this case, each partially generated SQL query (i.e. the FROM clause, which is discussed in the previous section) is duplicated for each unique expression, and the respective expression is appended to the query. An example illustrating this is shown in Figure 15.

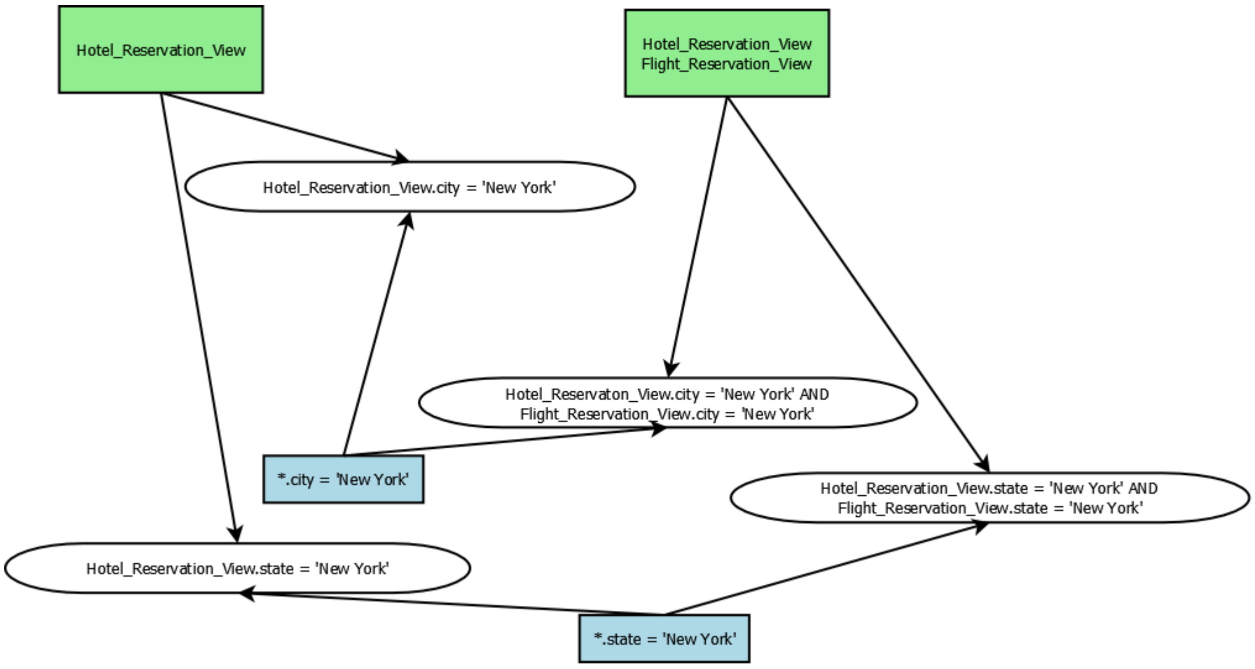

Hotel_Reservation_View
Hotel_Reservation_View
Flight_Reservation_View
Hotel_Reservation_View.city = 'New York'
Hotel_Reservaton_View.city = 'New York' AND
Flight_Reservation_View.city = 'New York'
*.city = 'New York'
Hotel_Reservation_View.state = 'New York' AND
Flight_Reservation_View.state = 'New York'
Hotel_Reservation_View.state = 'New York'
*.state = 'New York'

Figure 15: An example of the combination process when there are multiple sets of tables and multiple expressions. The green boxes represent unique sets of tables, and the blue boxes represent unique expressions. The white boxes are the resulting WHERE clauses generated by combining tables and expressions.

5.5 Combining the Partial SQL Queries

To form full SQL queries we need to combine the FROM clauses (the sets of tables generated from *Determining the Tables*) with the WHERE clauses (the sets of expression from *Determining the Expression*).

For each table combination in the set of tables, we match each expression in the set of expressions. This process forms complete SQL queries. To illustrate this process, a pseudo code algorithm is presented:

```
$full\_sql\_queries = \{\emptyset\}$
for $table\_combination$ in $tables\_set$:
    for $expression$ in $expressions\_set$:
```

$$full\_sql\_queries.add(table\_combination, expression)$$

5.6 Ranking the Queries

The queries are ranked in order from most likely to match the user's NLQ intention to least likely. This is done by relative weighting. There are two relative weights: one for the tables (the FROM clause) and one for the expression (the WHERE clause). The tables use a simple ranking based on the frequency calculation which is mentioned in the *Determining the Tables* section. The expression ranking is determined from the probability value found in the ontology to database mapping. These two rankings are combined to form an overall SQL query ranking which is eventually presented to the user.

5.7 SQL Generation Process Summary

The following is a summary of the entire SQL generation process. To see the same summary in graphical form, see Figure 10, located at the beginning of this *Methodology* section.

1. User enters a natural language query (NLQ)
2. User query is tokenized using the ontology; words that cannot be found in the ontology are discarded
3. Mapping look up on each valid token; if there is no mapping for the token, check the ontology node's children
4. Using the mappings for each token, determine which table(s) the user wants based on frequency and use this in the FROM clause
    a. This set of tables with the highest frequency is defined as the base set and is also defined to be most likely what the user intended
    b. Form new sets of tables by combining the base set of tables with each of the remaining tables; these sets will be secondary results with lower probabilities. Each of these resulting sets results in a unique SQL query. Calculate a probability value for each of these unique queries.
5. Using the mappings for each token, combine the expressions and use this as the WHERE clause

    a. If the base set of table(s) has more than one table, generate the join condition and append this to the WHERE clause
    b. If there are multiple mappings for a token (e.g. “new york”), then create a separate query for each set of mappings; its respective probability value will be used in the ranking of each query. Calculate an overall expression probability value for each of the unique queries.
6. Calculated a combined ranking/probability value for each resulting query; one of the probability values comes from the set of tables and the other comes from the set of expressions
7. Return a list of SQL queries to the user along with each of the queries’ respective probability values

6 Results and Discussion

In this section we present the UI that was developed, and we discuss the correctness and accuracy of some example user queries.

6.1 User Interface

A user interfaced was created so that the user can run their queries. However, before the user enters a query they must connect to the database and select the ontology file. We walk through each of these steps in the following subsections.

6.1.1 Connecting to the PostgreSQL Database

The system prompts the user to enter information about which PostgreSQL (PSQL) database the tool should connect to. The user must enter the: server address, PSQL user name, PSQL password, and the database name.

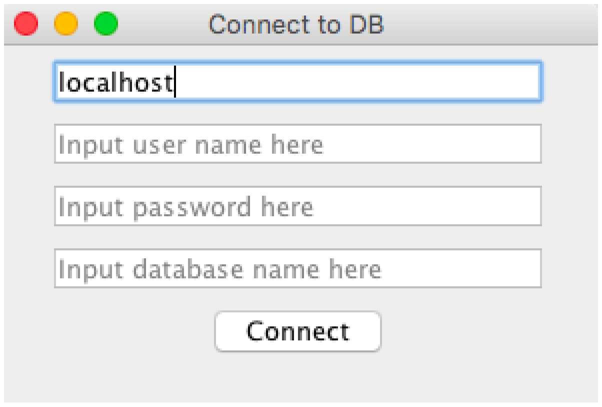
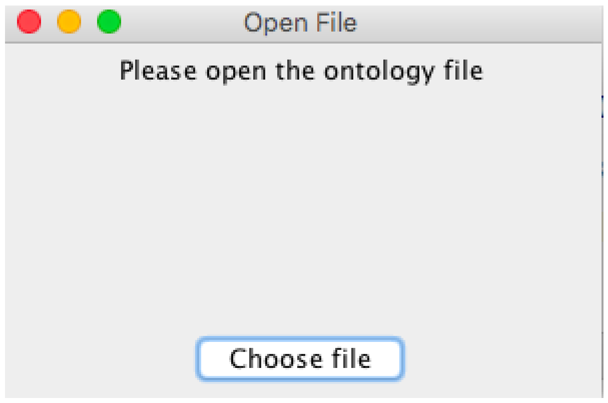


Figure 16: Connect to database screen. User must enter a server address, PSQL user name, PSQL password, and the database name to connect to.

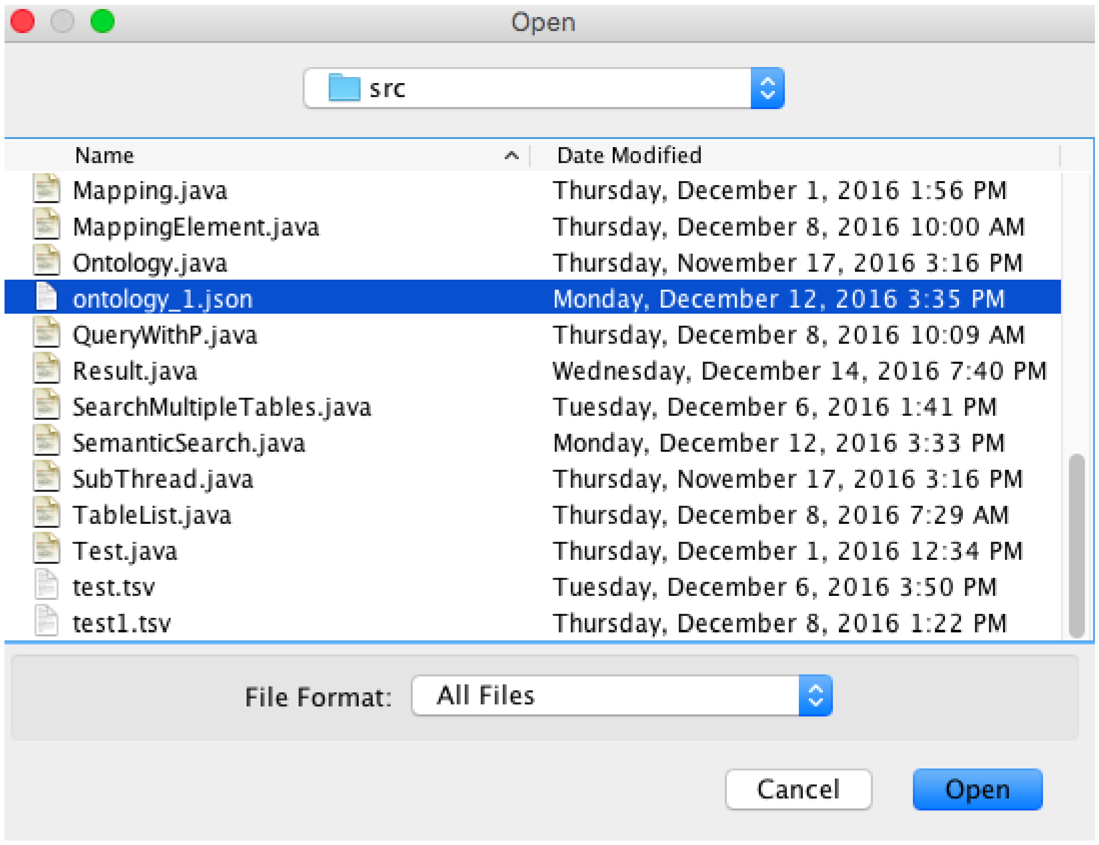
6.1.2 Selecting the Ontology

Next the user must select the ontology file which should be in JSON format. The structure for the file should follow the example file. Specifically, each JSON object should have a "name" and "children" property. This was discussed previously in the *Ontology Overview* section. Examples of the UI windows for choosing the ontology file can be found in Figure 17 and Figure 18.


Open File

Please open the ontology file

Figure 17: Select the ontology screen. The ontology should be a JSON formatted file.


Open

Name	Date Modified
Mapping.java	Thursday, December 1, 2016 1:56 PM
MappingElement.java	Thursday, December 8, 2016 10:00 AM
Ontology.java	Thursday, November 17, 2016 3:16 PM
ontology_1.json	Monday, December 12, 2016 3:35 PM
QueryWithP.java	Thursday, December 8, 2016 10:09 AM
Result.java	Wednesday, December 14, 2016 7:40 PM
SearchMultipleTables.java	Tuesday, December 6, 2016 1:41 PM
SemanticSearch.java	Monday, December 12, 2016 3:33 PM
SubThread.java	Thursday, November 17, 2016 3:16 PM
TableList.java	Thursday, December 8, 2016 7:29 AM
Test.java	Thursday, December 1, 2016 12:34 PM
test.tsv	Tuesday, December 6, 2016 3:50 PM
test1.tsv	Thursday, December 8, 2016 1:22 PM

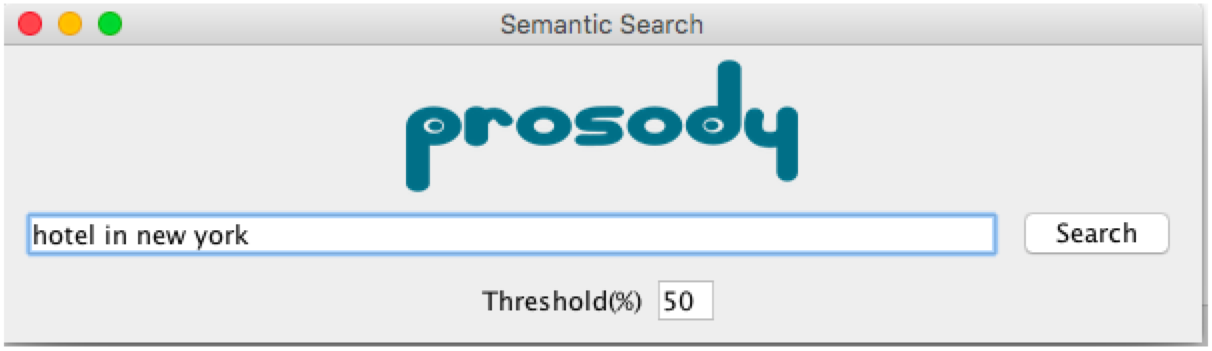
File Format: All Files

Figure 18: File selector screen. User can pick the ontology file.

If the JSON file could not be read in properly, then an error message will be displayed.

6.1.3 Entering a Natural Language Query and Setting the Threshold

Here the user can type in a query using natural language. Recall that the system only supports keywords in the ontology. Keywords that cannot be found in the ontology are discarded. The user can also set a threshold which is a percent value, in order to filter out low probability queries. The search window is shown in Figure 19.



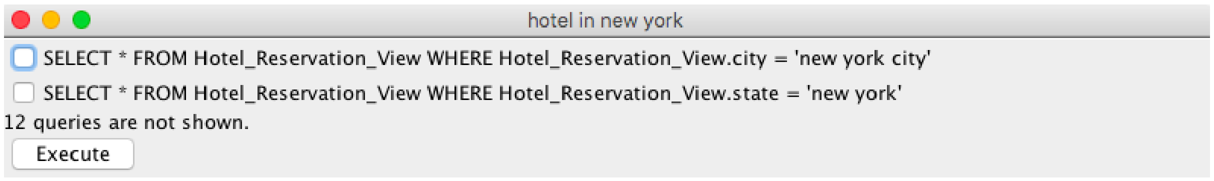
hotel in new york

Threshold(%) 50

Figure 19: User can enter a natural language query in the text box. A threshold can also be set to filter out results that have low probabilities.

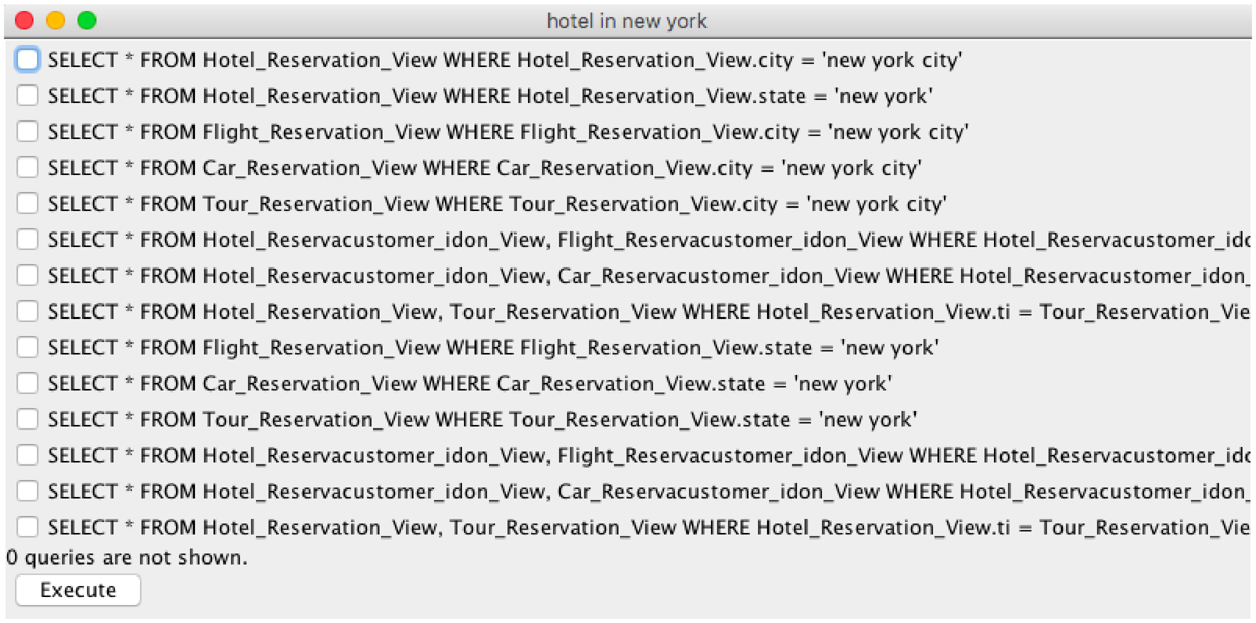
6.1.4 Choosing a SQL Query

Based on the user's NLQ, multiple SQL queries are returned with a relative ranking/probability. The user can choose which SQL query that they want to execute. Using the same query as before, "hotel in New York", an example SQL selection screen for a 50% threshold can be found in Figure 20, and the same query but with a threshold of 0% can be found in Figure 21.



```
☐ SELECT * FROM Hotel_Reservation_View WHERE Hotel_Reservation_View.city = 'new york city'
☐ SELECT * FROM Hotel_Reservation_View WHERE Hotel_Reservation_View.state = 'new york'
12 queries are not shown.
```

Figure 20: SQL selection screen for "hotel in New York" query. User can select which of the queries that they want to execute. This selection is based on a 50% threshold.


hotel in new york

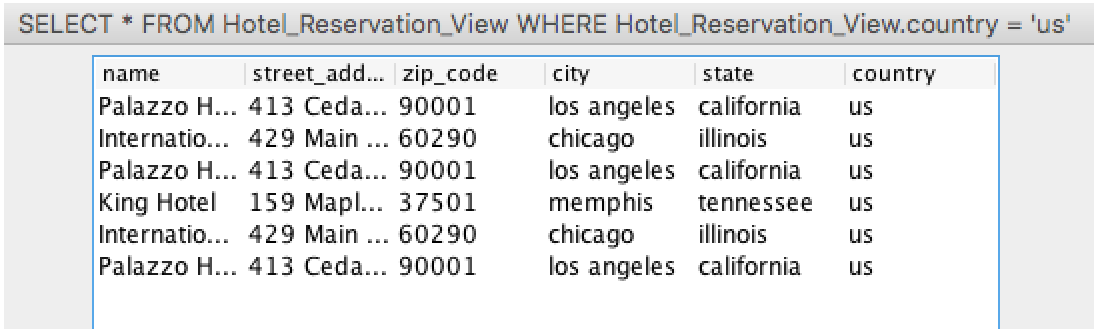
```
☐ SELECT * FROM Hotel_Reservation_View WHERE Hotel_Reservation_View.city = 'new york city'
☐ SELECT * FROM Hotel_Reservation_View WHERE Hotel_Reservation_View.state = 'new york'
☐ SELECT * FROM Flight_Reservation_View WHERE Flight_Reservation_View.city = 'new york city'
☐ SELECT * FROM Car_Reservation_View WHERE Car_Reservation_View.city = 'new york city'
☐ SELECT * FROM Tour_Reservation_View WHERE Tour_Reservation_View.city = 'new york city'
☐ SELECT * FROM Hotel_Reservacustomer_idon_View, Flight_Reservacustomer_idon_View WHERE Hotel_Reservacustomer_idc
☐ SELECT * FROM Hotel_Reservacustomer_idon_View, Car_Reservacustomer_idon_View WHERE Hotel_Reservacustomer_idon_
☐ SELECT * FROM Hotel_Reservation_View, Tour_Reservation_View WHERE Hotel_Reservation_View.ti = Tour_Reservation_Vie
☐ SELECT * FROM Flight_Reservation_View WHERE Flight_Reservation_View.state = 'new york'
☐ SELECT * FROM Car_Reservation_View WHERE Car_Reservation_View.state = 'new york'
☐ SELECT * FROM Tour_Reservation_View WHERE Tour_Reservation_View.state = 'new york'
☐ SELECT * FROM Hotel_Reservacustomer_idon_View, Flight_Reservacustomer_idon_View WHERE Hotel_Reservacustomer_idc
☐ SELECT * FROM Hotel_Reservacustomer_idon_View, Car_Reservacustomer_idon_View WHERE Hotel_Reservacustomer_idon_
☐ SELECT * FROM Hotel_Reservation_View, Tour_Reservation_View WHERE Hotel_Reservation_View.ti = Tour_Reservation_Vie
0 queries are not shown.
```

Figure 21: SQL selection screen for "hotel in New York" query. User can select which of the queries that they want to execute. This selection is based on a 0% threshold.

6.1.5 SQL Query Results

Lastly, the results from the SQL query are displayed to the user in a table format. Note that the database may not have any entries for certain queries, thus no results may be displayed. For example, the query "hotel in California during winter" displays 0 results. However if we leave out "winter", then the more generic query "hotel in California" displays 3 results.

An example of the result window is shown in Figure 22.


SELECT * FROM Hotel_Reservation_View WHERE Hotel_Reservation_View.country = 'us'

name	street_add...	zip_code	city	state	country
Palazzo H...	413 Ceda...	90001	los angeles	california	us
Internatio...	429 Main ...	60290	chicago	illinois	us
Palazzo H...	413 Ceda...	90001	los angeles	california	us
King Hotel	159 Mapl...	37501	memphis	tennessee	us
Internatio...	429 Main ...	60290	chicago	illinois	us
Palazzo H...	413 Ceda...	90001	los angeles	california	us

Figure 22: An example result window for the query "hotel in US". The results are displayed in table format, which reflects the database table format.

6.2 Comparing User Natural Language Queries and their SQL Results with User Intentions

In this section we will provide some example user queries, and we explain why each query would or would not match the user's intentions based on their natural language query. The examples can be found in Table 10.

Table 10: Comparison of example natural language queries with user intent. This table explains whether the system, as currently designed, would understand the user's intent. O indicates that the system would understand the user's intent and X indicates that the system would not understand the user's intent. -- indicates that the user's intentions are unclear.

Query	Understood user's intent?	Description
flight	O	Returns all customer trips that involve a flight
hotel in California	O	Returns all customer trips that involve a hotel in California
car rental in Massachusetts during winter	O	Returns all customer trips that involve a car rental in Massachusetts during the winter time
trip to Africa	O	Returns all customer trips to Africa. Since the user entered the generic term "trip" (as opposed to "hotel", "flight", etc.) the user will get customer trips that involve hotels, flights, car rentals, and/or tours.
flight to Panama	X	Panama is not in the ontology, therefore the keyword will be ignored. This results in the limited query "flight" which will return all customer trips involving a flight.

hotel in Brazil or Peru	X	Currently, the ontology does not understand the keyword "or". By default, all expressions are combined using "AND". Therefore, the resulting query will result in hotels that are located in both Brazil **and** Peru, which returns 0 results. As opposed to the expected result of hotels in Brazil **or** Peru.
sightseeing in New York	X	There is no keyword "sightseeing" in the ontology, and there is no table in the database that contains information about sightseeing. Therefore this word is discarded. Thus, the user will see all types of customer trips to New York.
flight from Worcester	X	The ontology does not understand the keyword "from". Therefore it is discard, and the query is simplified to "flight ~~from~~ Worcester". When looking at flights, the system currently assumes that user is looking for flights **to** a particular destination. Therefore, this query returns flights that are **to** Worcester rather than **from** Worcester. The notion of **to** and **from** could be implemented in future iterations of the project.

trip to Canada during winter in June	--	The user specified that they are looking for trips during winter in June. June is not during the winter time, therefore this query will always return 0 results. This is because the resulting expression will "AND" together the winter months with June, which have no overlap. However, this query's error is mostly the user's fault since it does not make much sense even from a natural language point of view.

7 Conclusion

We were successful in creating a tool which transforms a natural language user query into valid SQL queries. The tool uses a simplified ontology which defines the system understandable keywords. The ontology also enables “is a” (i.e. parent-child) relationships between keywords for a better understanding of the natural language. The ontology to database mapping enables the tool to understand how ontology keywords are directly related to the database schema. Using these two concepts, the tool is able to determine the relevant database tables, column names, and expressions. From here, SQL queries are generated along with respective probabilities.

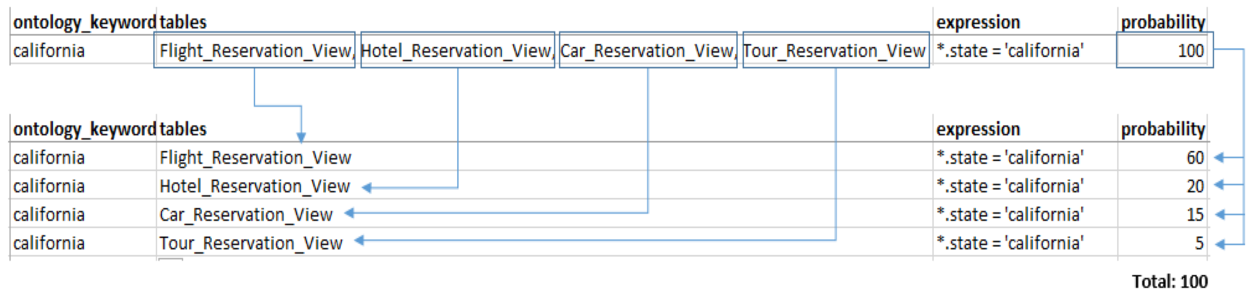
Additionally, we believe that the tool is scalable in multiple aspects. For example, new keywords can easily be added to the system. The keyword simply needs to be inserted into the ontology, and then a new ontology to database mapping entry needs to be added. The tool was also designed to be useable independent of the database schema. If the schema changes, then a new mapping needs to be created, but ideally the tool itself should not need to change. However, the tool has not been tested on other schemas due to time constraints, so it is possible that it may need some refinement in this aspect.

8 Recommendations

This section discusses possible improvements or additional features that could be added to the system/tool.

8.1 Improving the Customizability of Keyword Probability

The keyword based probability value is contained within the ontology to database mapping. Currently, the keyword probability is defined for a set of tables, as shown in the top portion in Figure 23. However, the probably could be redefined for each table that it maps to, as shown in the bottom portion of Figure 23.



ontology_keyword	tables	expression	probability
california	Flight_Reservation_View, Hotel_Reservation_View, Car_Reservation_View, Tour_Reservation_View	*.state = 'california'	100

ontology_keyword	tables	expression	probability
california	Flight_Reservation_View	*.state = 'california'	60
california	Hotel_Reservation_View	*.state = 'california'	20
california	Car_Reservation_View	*.state = 'california'	15
california	Tour_Reservation_View	*.state = 'california'	5

Total: 100

Figure 23: An example of splitting the probability value. This split means that the probability can be associated with each table independently, allowing for a higher level of customization.

This change allows the probability to be customized for each of the keyword's related tables. Additionally, this probability value could be modified historically based on the type of travel that a user tends to associate with a particular keyword. For example, if the vast majority of users tend to search for "flights to California" (as opposed to hotels, tours, etc.), then the probability value for the California to flight mapping can be increased.

8.2 Expansion of Ontology

The system currently uses a simplified version of the OWL standards for the ontology. Specifically, our version of the ontology only uses "is a" relationships, objects, and classes. However, ontologies are capable of more, such as complex relationships and attributes. For example, ontology creators can define their own relationships between nodes. See the "eats" relationship from the animal ontology in Figure 1 for a simple example. Additionally, values for attributes can be assigned to ontology objects and classes.   See the OWL Web Ontology Reference Guide for more details on the capabilities of OWL ontologies: https://www.w3.org/TR/owl-ref/.

If the ontology were to be expanded with more detail and increased complexity, then this would enable the system to take advantage of more properties and relationships, and thus, it could provide even more accurate results.

Some specific ideas for ontology keyword expansion are presented in Table 10; note the queries where the system did not understand the user's intent.

9 References

Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., & Stein, L. A. (2014, February 10). *OWL Web Ontology Language*. Retrieved from W3: https://www.w3.org/TR/owl-ref/

DB-Engines Ranking. (2016, December). Retrieved from DB-Engines: http://db-engines.com/en/ranking

Garshol, L. M. (2004, October 26). *Metadata? Thesauri? Taxonomies? Topic Maps!* Retrieved from Ontopia: http://www.ontopia.net/topicmaps/materials/tm-vs-thesauri.html#N773

PostgreSQL: PostgreSQL Press FAQ. (n.d.). Retrieved from PostgreSQL: https://www.postgresql.org/about/press/faq/

Saha, D., Floratou, A., Sankaranarayanan, K., Minhas, U. F., Mittal, A. R., & Ozcan, F. (2016). ATHENA: An Ontology-Driven System for Natural Language Querying over Relational Data Stores. *VLDB*, 1209-1220.

White, S. (2005, August). *Better computational descriptions of science*. Retrieved from Scientific Computing World: https://www.scientific-computing.com/feature/better-computational-descriptions-science