Ontology-based natural language interface to public security population database

Xiangwu Ding, Hao Liu*

College of Computer Science and Technology, Donghua University, Shanghai 201620, China

870350571@qq.com

Abstract. Natural Language Interface to Database (NLIDB) could convert natural language queries into SQL automatically, which has been extensively studied. However, how to apply NLIDB to the public security population database (PSP-DB) remains an open problem due to the challenges to utilize domain knowledge and generate complex queries involving multiple tables. To tackle these problems, this paper proposes an ontology-based NLIDB approach combining with public security population ontology (PSP-Ontology) and syntactic analysis. Its key idea includes: (1) constructing an ontology from the schema of PSP-DB and extending it with synonym expansion; and (2) proposing an association path processing algorithm to handle multi-table connection path in SQL generation. We have evaluated the approach on the population database from Shanghai Public Security Bureau. The results show that the PSP-Ontology and association path processing algorithms could alleviate these two challenges and improve the accuracy of SQL translation effectively.

Keywords: ontology; natural language interface to database; public security population database;

1 Introduction

With the development of information technology, database is widely used in government departments, commerce, academia and other fields. Currently, the population database of Shanghai Public Security Bureau has accumulated a large amount of data. There are up to 1143 tables, the relationship between which is very complex. In addition, users in public security department need to query the database flexibly in dealing with daily business. The traditional way to query the database is to write specific SQL statements. This method requires users to be highly familiar with the database schema and the usage of SQL statements, which is not practical and efficient for end users. Therefore, natural language interface to database (NLIDB), a method that could convert natural language queries into SQL automatically, is promising to bring great convenience for users in public security department.

The research on NLIDB is mainly divided into deep learning based approaches [1][2] and pipeline based approaches [3]. Approaches based on deep learning usually take the query statement and related database tables as the model input, and generate

V. Escudero et al. (Eds.): BADS 2022, AHCS 11, pp. 348–365, 2023.
https://doi.org/10.2991/978-94-6463-102-9_38

SQL statements through neural network models like Sequence-to-Sequence [1]. However, they require tremendous labelled data for training and lack of capabilities to generate multi-table complex SQL statements. The pipeline based approaches are similar to a working assembly line in production workshops, which consist of a series of modular processing stages to complete the transformation. The advantage of pipeline based approaches is that they do not need labelled training data, and more flexible to deal with multi-table complex queries. Considering the lack of labelled training data in the field of public security population and the need for multi-table complex query, this paper designs and implements natural language query to the public security population database based on pipeline.

Unfortunately, although pipeline based NLIDB has drawn extensive research attention to facilitate non-professional users to interact with database, it is still challenging to apply it to the public security population database (PSP-DB). Firstly, general NLIDB techniques are unaware of domain knowledge, thus could not recognize domain specific natural language expressions both in user queries and database table names. Secondly, the complex relationship between tables in PSP-DB brings difficulties to generate complex queries involving multiple tables.

To resolve the two problems, we propose an ontology-based NLIDB approach to translate natural language queries into SQL statements. The ontology of the public security population is first constructed by parsing and mapping the schema of the PSP-DB into Ontology Web Language (OWL). After that, the natural language queries are transformed into an intermediate language using syntactic analysis with the ontology. Finally, we propose an association path processing algorithm to generate SQL statements from intermediate language. We have evaluated the approach on the population database from Shanghai Public Security Bureau. As the results in Section 4 show, the ontology based NLIDB achieves 83.3%, 68.8% and 51.9% precision rate of the translation of single table query, multi-table query and complex query respectively. This paper makes the following contributions:

(1) The public security population domain ontology is established to help improve the accuracy of subsequent syntactic analysis and SQL statement generation.

(2) The semantic disambiguation of synonyms is carried out in combination with the synonym forest so that the interface can accept the flexible natural language input of users.

(3) The association path algorithm is proposed, which is combined with the Steiner tree optimization algorithm to better generate multi-table complex query statements.

2 Related Work

The natural language interface to database (NLIDB) transforms natural language into a machine language that database could parse and execute, e.g. SQL [4]. Early researchers used natural language processing techniques such as pattern matching and keyword matching to perform the transformation and experimented them in specific systems, such as the American Baseball League database query system (BASEBALL system). However, such systems rely on template libraries and only support queries in specific

formats. Subsequent researchers proposed knowledge models such as extended transfer network, conceptual dependency theory, and grid grammar to assist computers in understanding natural language. Thanks to these technologies, a large number of NLIDB customized for specific databases emerged during this period, e.g. LUNAR, LADDER, PLANES. With the rapid development of natural language processing (NLP), multiple NLP techniques were applied in the development of NLIDB. NaLIR [5] uses the existing parser to obtain the dependency parsing tree of a given query. It then maps the nodes of the parse tree to SQL components. When the natural language query is not clear, NaLIR relies on user interaction to perform disambiguation, but too much interactions increase the burden on users. ATHENA system [6] takes the ontology query language (OQL) as the intermediate languguage from natural language to SQL statement. It establishes the ontology knowledge base and analyses user queries in combination with syntax rules. The idea is only effective in the simple query, because there will be semantic mismatch between ontology and semantics of complex query, resulting in the decline of accuracy. After that, the same research team proposed ATHENA++ system [7], which propose a nested query classifier on the basis of ATHENA system to deal with multi-layer complex nested queries. However, ATHENA++ is hard to be migrated to other domains and is still difficult to deal with large-scale data.

In recent years, a large number of researchers have applied deep learning to the research of NLIDB. Encoder-decoder based on recurrent neural network (RNN) is a neural network architecture widely used in NLIDB. The natural language query is analysed by encoder-decoder, and finally transformed into SQL statement. Zhong et al. [8] proposed the Seq2SQL model. In the decoding process, the slot value is filled based on the fixed template, and the SQL generation is divided into three subtasks to predict the AGG, SELCOL, and WHERE clauses respectively. Xu et al. [9] proposed SQLNET on the basis of Seq2SQL, further refined the SQL generation subtask and divided the WHERE clause generation into four interdependent subtasks. However, due to the lack of domain knowledge, approaches based on deep learning perform poor in transforming query statements of specific domain databases, and is limited to single table query and specific query forms. By integrating domain knowledge, this paper improves the flexibility of user input queries and the accuracy of transformation.

3 System Architecture

The overall architecture of the ontology-based NLIDB for public security domain is shown in Fig.1. The system constructs knowledge base and domain ontology by crawling public data and database table structure and expands the ontology with two expansion strategies. After the police user enters a natural language query, the system first parses the query statement to generate the intermediate language, including query preprocessing, query targets generation and query conditions generation. Then the association path processing algorithm is applied to deal with multi-table association, and the intermediate language is transformed into a complete SQL statement. Domain ontology and knowledge base participate in the whole process and play a role in assisting intermediate language transformation and SQL statement generation.

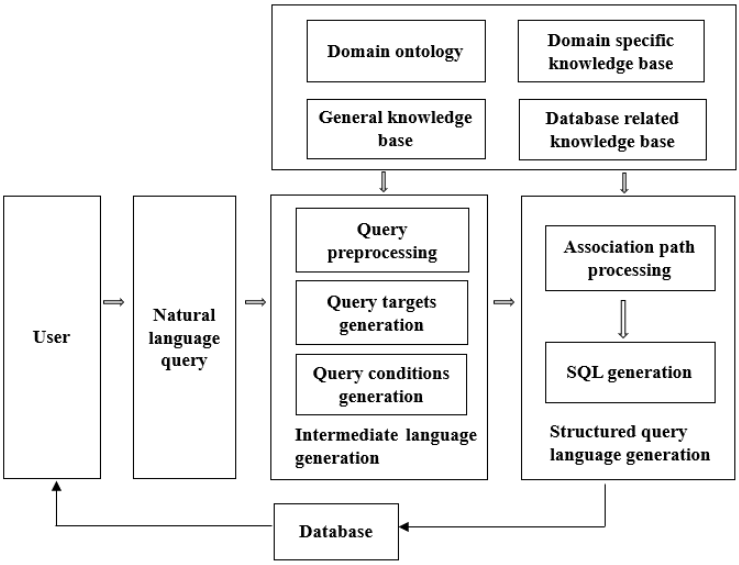

Fig. 1. System architecture of ontology-based NLIDB

3.1 Knowledge base and domain ontology

To improve the accuracy of intermediate language and final SQL statement generation, domain knowledge and database related information are needed. This paper crawls the public data of the Internet, constructs the domain ontology and knowledge base combined with the analysis of the table schema of the public security population database, and expands the domain ontology with the synonym forest. This section presents the procedure of knowledge base construction, domain ontology construction and domain ontology expansion.

3.1.1 Knowledge base construction.

Without semantics and information related to the database, it is difficult to analysis and transform natural language queries. Thus, we construct the knowledge base to facilitate the transformation process, dictionary in which plays a vital role in the transformation from natural language into SQL statements. The knowledge base mainly includes three parts: general knowledge base, domain specific knowledge base and database related knowledge base. Among them, the general knowledge base contains domain and database irrelevant words types, e.g. the logical word "as well as" corresponds to "AND", and the aggregate word "average" corresponds to "AVG"; The domain specific knowledge base contains domain-related words, which are collected by crawlers, e.g. Shanghai street names; Database related knowledge base contains database table information, including primary key, foreign key, etc. This paper constructs a general knowledge base and domain specific knowledge base through crawler and builds a database-related knowledge base through the analysis of database table schema.

3.1.2 Domain ontology construction.

The goal of domain ontology is to capture and structure the domain knowledge of a specific domain and provide a general understanding of the domain. In this paper, Ontology Web Language (OWL) [10] is used to describe the ontology model. E-R model is a tool to describe entities and relationships between entities in the real world. It can be extracted from the schema of relational database. Relevant elements in the database E-R model are transformed into concept classes, properties and individuals in the ontology model. The following E-R model describes the conceptual structure of relational database:

$$ER = (E, R),$$

$$E = (A, K, C),$$

$$A = (T, D, R),$$

$$R = (E1, E2, FK)$$

where E represents entities, A represents attributes, R represents relationships, T represents attribute value types, D represents domains, K represents primary keys, FK represents foreign keys, and C represents constraints. The process of generating public security population domain ontology from E-R model is as follows:

(1) Define OWL files. An ontology model is transformed from an E-R model. It is necessary to define an independent OWL file of public security population domain ontology to represent the model.

$$ER\_Model(ER) \rightarrow Ontology(OWL)$$

(2) Extract concept classes. A concept class represents a fact concept in specific domain. The corresponding mapping rules convert the entities in the E-R model into concept classes in the ontology and map them into owl: Class in OWL files. For example, the express entity and student entity correspond to the express class and the student class respectively in OWL.

$$ER\_Model(E) \rightarrow Ontology(owl{:}\ Class)$$

(3) Extract data attributes. Data attributes are the characteristics of specific domain concepts. The mapping rules convert the attributes of entities in E-R model into the data attributes of corresponding ontology concept classes. In OWL, rdfs: Domain specifies the subject resource and rdfs: Range specifies the object resource. The two types of entities associated with the association are mapped to rdfs: Domain and rdfs: Range.

$$ER\_Model(A) \rightarrow Ontology(owl{:}\ DatatypeProperty, rdfs{:}\ Domain, rdfs{:}\ Range)$$

(4) Extract object properties. Object attributes are the relationship between domain concepts. The corresponding mapping rule is to transform the relationship between

entities in E-R model into object properties in ontology and map them to owl: Object Property.

$$ER\_Model(R) \rightarrow Ontology(owl: ObjectProperty)$$

(5) Extract data instances. Since the stored data in the database will appear in natural language query statements, we need to map these data to individuals.

$$Data(ER) \rightarrow Ontology(Individuals)$$

This paper applies D2RQ tool(http://d2rq.org/) to implement the above conversion rules, the constructed public security population domain ontology is shown in the following Fig.2.

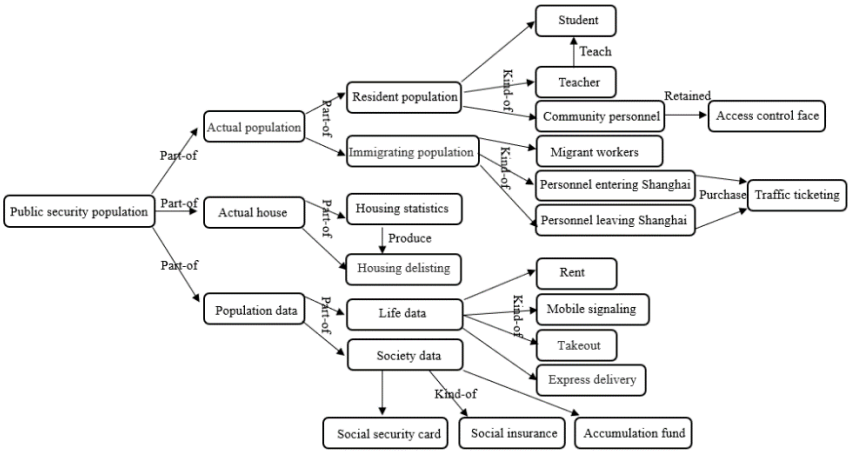

Fig. 2. Public security population domain ontology

Through the above steps and rules, the E-R model of the public security population domain database is transformed into a preliminary OWL ontology model. In addition, we need to enrich the ontology model through ontology expansion to facilitate the generation of intermediate languages and final SQL statements. Ontology expansion includes the integration of public security specific knowledge information such as related concepts and synonymous expressions into the ontology model.

3.1.3 Domain ontology expansion.

Domain ontology expansion helps deal with the flexible query statements of users. This paper designs and implements two domain ontology expansion strategies:

(1) Synonym expansion. As users are not familiar with the database table schema, synonyms and approximate expressions are common in natural query sentences. Therefore, this paper applies the synonym forest (extended version) proposed by Harbin Industrial University (https://www.ltp-cloud.com) to find the synonyms in the descriptions of concept classes or data attribute of the ontology and add these synonyms to the corresponding descriptions. The synonym forest contains 77343 words in total. All

words in the forest are in a tree hierarchy, which is divided into three categories: large, medium and small, with 12 large categories, 97 medium categories and 1400 subcategories. The subcategories are further divided into atomic word groups according to the word meaning. The complete forest tree includes five layers of category, including the root node layer. Combined with the word forest structure and coding form, this paper defines the semantic similarity between the two words $C1$ and $C2$ as:

$$sim(C1, C2) = m * \cos\left(\frac{n\pi}{180}\right) * \left(\frac{n-k+1}{n}\right) \quad (1)$$

where m is the weight of the layer of the word forest coding, which is assigned manually as 0.6, 0.7, 0.91, and 0.97 from the second level to the fifth level, n is the total number of words in the layer where $C1$ and $C2$ are located, and k is the distance between the two words, which is used to adjust the value of semantic similarity to make it fall into the interval [0,1]. If the semantic similarity between $C1$ and $C2$ exceeds the preset threshold of 0.9, it is a similar word. This paper uses the above formula to find all symptoms of words in descriptions of concept classes or data attribute and adds them to the owl: Synonym field of the corresponding ontology element.

(2) Default attribute expansion. The user may not give the required attribute for convenience, thus the concept class needs to be equipped with a default query attribute. This paper utilized the database related knowledge base to expand the default attribute field for the concept class in the ontology model and add it to the owl: Datatype Property field of each concept class. For example, the student entity is equipped with the default attribute field "student's ID number". When the query statement has only the word "student", it can be inferred that the default query attribute is "student. student's ID number ".

3.2 Intermediate language generation

This section first gives the grammatical definition of query statements and the definition of intermediate language. After that, we present the process of converting natural language query statements into an intermediate language.

3.2.1 Query syntax definition.

This paper only focuses on a single round of natural language query statement, so the syntax is relatively simple and belongs to context-free grammar. The query statement complies with the following grammar rules:

$$< natural\ language\ query\ statement > \rightarrow\ [< query\ word >][< query\ conditions >] < query\ targets >,$$

$$< query\ word > \rightarrow\ X,$$

$$< query\ conditions > \rightarrow\ [< grouping\ phrase >] < condition\ phrase > [< grouping\ phrase >]\{[< grouping\ phrase >][L] < condition\ phrase > [< grouping\ phrase >]\},$$

$$< grouping\ phrase > \rightarrow\ G[E|P|EP],$$

$$< condition\ phrase >$$
$$\rightarrow V|POV|EPOV|EOV|FPOV|EEOV|FP|PF|EFPOV|POVP|EPOVP|EOVP\ |FPOVP|EEOVP|EFPOVP\ ,$$

$$< query\ targets > \rightarrow < target\ phrase > \{[L] < target\ phrase >\},$$

$$< target\ phrase > \rightarrow\ E|EE|EF|EFP|EP|FE|FP|P|PF$$

where X represents query verbs, L represents logical words, G represents grouping words, E represents entity words, P represents attribute words, V represents attribute value words, O represents relational words, F represents function words, and Z may also be used to represent auxiliary words.

3.2.2 Intermediate language definition.

The intermediate language of the system consists of sentence pattern array (*SPA*), sentence object array (*SOA*), entity array (*EA*), query target phrase array (*QTPA*), query condition phrase array (*QCPA*), query targets array (*QTA*), query conditions array (*QCA*) and group array (*GA*). Once the query statement is automatically segmented, the standard word (A synonymous representation is a metadata representation stored in a database) of the word is found in combination with the domain ontology and knowledge base, the corresponding word type are stored in the *SPA*, and the standard word is stored in the *SOA*. The *EA* stores the entity table involved in the query statement. The *QTPA* stores the description of the query targets in the user statement, which is used to generate the *QTA*. The *QCPA* stores the description of the query conditions in the user statement and is used to generate the *QCA*. The *QTA* stores multiple query targets extracted and identified. The *QCA* stores the conditional constraints that the query needs to meet and the logical relationship between the conditional constraints. The *GA* stores the contents required to generate the GROUP BY clause in the SQL statement.

3.2.3 Query statement pre-processing.

After the user enters the query statement, the query statement is pre-processed, including Chinese Word Segmentation, Generation of Sentence Pattern Array and Sentence Object Array, Target Phrase Extraction and Condition Phrase Extraction.

(1) Chinese Word Segmentation. In order to ensure the effectiveness of word segmentation, this paper uses Jieba, a widely used Chinese word segmentation tool(https://pypi.org/project/jieba/), combined with the dictionary in domain specific knowledge base to segment the sentence. The word segmentation results are shown as ['query', 'number', 'is', '2202523', 'name'].

(2) Generation of Sentence Pattern Array and Sentence Object Array. The *SPA* is the basis for subsequent query target and query condition analysis. The generation process is as follows:

① Match words to entities in the domain ontology model. Words can be matched to the names of concept classes, data attributes and object attributes in the ontology model, and can also be matched to the entity synonymous expression and default attribute fields after the expansion of the ontology model.

② Search the domain ontology and knowledge base to find the standard word of the word.

③ Add the matched parts of speech into the sentence pattern array, which may be word categories such as concept class, data attribute, object attribute and conjunction.

For example, the *SPA* corresponding to the above word segmentation results ['query', 'number', 'is', '2202523', 'name'] is $[XPOVZP]$. In this example, the standard word "number" in the database is "ID number", so the *SOA* is ['query', ' ID number ', 'is','2202526', 'name'].

(3) Target Phrase Extraction. The natural language input by the user includes query verbs, query target phrase and query condition phrase according to grammar rules. Although the position of the query targets in the query sentence is affected by the sentence pattern, there exists some patterns in expression of the target phrase. Therefore, this paper uses rule matching to extract the query target phrase. The extraction process is as follows: sort the target phrase rules in the grammar from large to small according to the length, in the other words, arrange them in the order of $EFP|EE|EF|EP|FE|FP|PF|E|P$, and match them with the word segmentation string from back to front. If the match is successful, the word is stored in the *QTPA*, and the word is deleted from the back of the *SPA*. If a stop sign, comma or some logical connectives appear, it indicates that there are multiple query targets, and the connecting characters are deleted from the word segmentation string. Recycle the above matching steps for the remaining word segmentation strings until the division of the whole word segmentation string is completed. If the query target cannot be found at last, an error will be reported. The generated query target phrase is stored in the *QTPA* for subsequent query targets generation.

(4) Condition Phrase Extraction. The extraction procedure of the query condition phrase is the same as that of the query target phrase. The condition phrase rules in the grammar are sorted from large to small according to the length, and the extraction process is consistent with that of the above query target. Query conditions can be left blank. The generated query condition phrase is stored in the *QCPA* for subsequent query condition generation.

3.2.4 Query targets generation.

The query target phrase identified in the pre-processing stage does not conform to the semantic expression in the database. Next, the *QTA* is generated as the intermediate language of the query targets to facilitate the splicing of query targets in subsequent SQL generation. Query targets can be divided into attribute target, entity target, full-name target and aggregation target. These query targets and their generation are introduced below.

(1) Attribute target. The query target type corresponding to the attribute target is *P*. If "name" is the attribute name, it belongs to the attribute target. This kind of query target is common. There is only attribute name but no entity name. The system needs to find the entity table corresponding to this attribute. The algorithm idea is as follows: store the entity table where the query condition is located, such as the "student" entity table, into the entity array, then take out all the entity tables in the entity array, find all

the attributes $totalP$ in combination with the domain ontology, and intersect the $totalP$ with the "name", if $P \cap totalP \neq \Phi$. The entity is returned and stored in the entity array.

(2) Entity target. The query target type corresponding to the entity target is E. If "student" is an entity target, there is a corresponding student table in the database. For the processing of such query targets, the default attribute "student's ID number" of the entity can be found directly on the domain ontology.

(3) Full-name target. The query target type corresponding to full-name target is $EP|EZP$. For example, "student's name" includes entity name and attribute name, which belongs to the full-name target. The standard words *stanE* and *stanP* corresponding to E and P can be found, and directly return the query target *stanE.stanP* and store it in the *QTA*, and store the entity *stanE* in the *EA* at the same time.

(4) Aggregation target. The query target type corresponding to the aggregation target is $FP|PF|EF|FE|EFP$. If "maximum salary" contains aggregation words, it belongs to the aggregation target. The main feature of this target phrase is the existence of aggregation function F. The processing methods of FP and PF are the same. First separate F, process P according to the attribute target, and then store it in the *QTA* in the form of $F(E.P)$.

3.2.5 Query conditions generation.

Next, the *QCA* will be generated and used as the intermediate language of query conditions to facilitate the transformation of subsequent query conditions. Query conditions can be divided into nested and non-nested condition. Non-nested condition can be subdivided into value condition, value-name condition, aggregation condition and grouping condition.

(1) Value condition. The query condition type corresponding to the value condition is V. For example, "apple juice" in "find the unit price of apple juice" is the value of the attribute column, which belongs to the value condition. The ambiguity problem should also be considered in the analysis algorithm of value condition. The processing algorithm of this kind of query condition is the same as that of attribute target.

(2) Value-name condition. The query condition type corresponding to the value-name condition is $POV|EPOV|EEOV|EOV$. For example, "find products with unit price higher than $20 ", where "unit price higher than $20 " contains attribute column name, relational word and attribute value. For the analysis of such conditions, the element before the relational word can be used to exclude the ambiguity of the entity corresponding to V. for $EPOV|EEOV|EOV$, the first entity E can be directly used as the query condition entity. For POV, you still need to use the *EA* to determine the conditional entity. Add the $EPOV$ of the complement component to the *QCA*.

(3) Aggregation condition. The query condition type corresponding to the aggregation condition is $EFPOV|FPOV|FP|PF$. For example, "find the product with the highest unit price", where "the highest unit price" needs to use the aggregation function. The aggregation condition can be processed by sub-query. Taking PF as an example, the algorithm idea is taking out the conditional phrase and finding the entity E corresponding to P through the domain ontology. If there is ambiguity, use the *EA* disambiguation

and find the aggregation function corresponding to F in combination with the general knowledge base. Add $F(E.P)$ to the QCA, and add entity E to the EA at the same time.

(4) Grouping condition. The query condition type corresponding to the grouping condition is $G[E|P|EP]$. For example, "the highest unit price of each type of product", afterword segmentation, "each (G) type (P)" is added to the GA, and "the highest (F) unit price of product (E) (P)" is added to the QTA. Here, the $E.P$ in the grouping segment needs to be matched with each item in the QTA. If there is no $E.P$ in the QTA, it will be added to this array.

(5) Nested condition. This type of condition phrase is the condition of related sub-query implicit in the query statement. To understand this sentence pattern, we must first determine the query expression of sub-query. It is easy to recognize the existence of OVP in the form of nested query. Maybe the front of O is $P|EP|EE$, but they can all get the form of EP through the domain ontology. Here, if "P" in EP and "P" in OVP are the same attribute, then the VP component in OVP is a nested condition, V is the sub-query condition and P is the sub-query target. For example, "find products whose unit price is higher than the unit price of apple juice", we can get "higher than (O) apple juice (V) unit price (P)". For nested sub-query "apple juice (V) unit price (P)", we can use the previous non-nested query objectives and query conditions to generate ideas.

3.3 Structured query language generation

This section introduces the association path processing algorithm and its optimization algorithm and describes the process of transforming intermediate language into complete SQL statements.

3.3.1 Association path processing.

In the process of transformation, the most important step is to deal with the problem of multi-table connection path. If the table where the query target is located is the same as the table where the query conditions are located, that is single table query. If it is different, it is multi-table query. The form of the association path between multiple tables needs to be discussed in detail.

Def.1 Direct Association Path. Two tables are directly associated through the primary key and foreign key.

Def.2 Indirect Association Paths. Two tables are connected through an intermediate table, which can be summarized as special cases of complex association paths.

Def.3 Complex Association Paths. The complex connections between three or more tables.

This paper will do general research on the association path, that is, considering the processing of complex association paths. General natural query statements do not contain hidden Association path information. The natural language query interface needs to automatically find out the relationship between tables and generate the correct association path. The general form of the Complex Association Paths is shown in Fig.3:

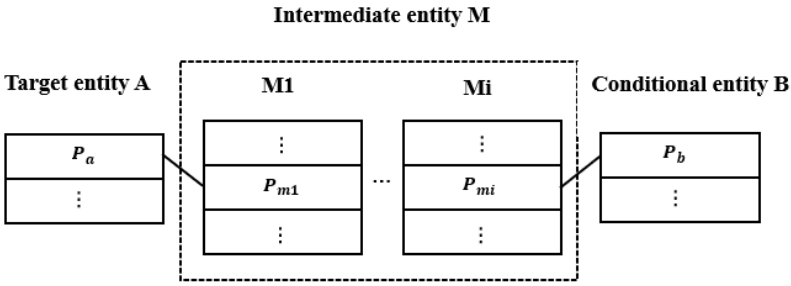

Fig. 3. The general form of complex association path

As shown in Fig.3, when the query target of the entity table A is different from the query condition of the entity table B, the two tables A and B need to be associated through intermediate entities. The intermediate associated entity table may be one, or it may contain multiple in complex cases. Given the above complex situations, this paper proposes an association path algorithm to solve the association path problem. The specific algorithm is shown in algorithm 1.

Algorithm 1: Association Path Processing

Input: TableList = { $T1, T2,\ldots, Tn$ }

Output: JoinList

1) Initialize JoinList;
2) Initialize database incidence matrix;
3) **if** len(TableList)<=1 **then**
4) **return** JoinList;
5) **else**
6) Set the first node $T1$ in the TableList as the reference node;
7) **Foreach** Tr in { $T2,\ldots, Tn$ } **do**
8) In the incidence matrix, the BFS search algorithm is used to find the shortest path T_path between $T1$ and Tr;
9) **if** the node in the T_path does not exist in the JoinList **then**
10) add T_path;
11) **else**
12) skip and do not add repeatedly;
13) **return** JoinList;

The algorithm process of generating the association path described above takes the Actual Population table and the Education Committee table as an example, TableList = {syrk, jw}. The Actual Population table and the Education Committee table cannot be directly connected through the primary key and foreign key. The association path algorithm automatically finds the intermediate table connected by the two tables, and then generates JoinList = {syrk, zjb, jw}. The tables in the JoinList are arranged in order according to the connection relationship and finally returned. Because the algorithm takes the first node in the data table set as the reference node, the selection of the

reference node may affect the generation of the final association path. In addition, repeated association paths may be generated through the above algorithm, so it is regarded as a local optimal scheme.

In this paper, the above association path algorithm is globally optimized in combination with Steiner tree [11], and the search of its association path is regarded as the generation of Steiner tree. The main idea is to generate a tree containing all nodes of the data table set TableList in the graph represented by the Ontology Graph. the nodes in the tree need to meet the requirements of the shortest path between two, and the generated tree needs to achieve the minimum overhead.

Def.4 Ontology Graph. It belongs to the concept class in the same ontology, and the corresponding data table has the association relationship. It is a graph constructed by the association relationship in relational database.

The Ontology Graph is formally defined as follows:

$$G = (D, E, W)$$

Where G is the Ontology Graph; D is the set of data tables contained, that is, the set of nodes in the graph; E is the edge set of the graph, corresponding to the primary key and foreign key association in the relational database; W is the weight of each edge and is set to 1 by default. The specific algorithm is shown in algorithm 2.

Algorithm 2: Path Optimization Based on Steiner Tree

Input: TableList = $\{T1,T2,\ldots,Tn\}$

Output: JoinList (Steiner tree T_s)

1) Initialize JoinList;
2) Initialize Ontology Graph G;
3) The Ontology Graph G and the data table set TableList generate a complete graph $G1$ ($D1$, $E1$, $E1$), where $D1$ = TableList, $E1$ is the associated edge set;
4) Generate a minimum spanning tree T_m in graph $G1$;
5) Replace the edge in the minimum spanning tree T_m with the corresponding shortest path in the graph G to obtain the subgraph G_k of the graph G; (if there are multiple shortest paths, select one at random)
6) The minimum spanning tree T_k is obtained from subgraph G_k;
7) Delete unnecessary nodes in the minimum spanning tree T_k to obtain the Steiner tree T_s. The leaf node set of T_s is equivalent to the node set TableList;

3.3.2 SQL generation.

There is a semantic gap between natural language and SQL statements. There will be great challenges in direct transformation, and semantic transition needs to be carried out in the form of intermediate language. The components of the intermediate language formed by the above series of processes are: sentence pattern array (*SPA*), sentence object array (*SOA*), entity array (*EA*), query targets array (*QTA*), query conditions array

(*QCA*), group array (*GA*), query target phrase array (*QTPA*) and query condition phrase array (*QCPA*). The SELECT clause is transformed by the *QTA*, the FROM clause is transformed by the *EA*, the WHERE clause is transformed by the *QCA*, and the GROUP BY clause is transformed by the *GA*. After splicing, a complete SQL statement is generated. The SQL generation process is as follows:

(1) Initialize the query target string, add 'SELECT', cycle through the *QTA* and add each target in the array in turn. Each target is separated by ',', and the last target is not added with ','; Initialize the entity string, add 'FROM', loop through the *EA*, and add the entities in the array in turn. Each entity is separated by ',', splicing the query target string and entity string. The generated results are as follows: SELECT [aggregate function] entity. attribute name [, entity. attribute name] FROM entity.

(2) Initialize the query condition string, add 'WHERE', cycle through the *QCA*, and add the logical relationship between the query conditions and conditions in the array in turn. At this time, you need to judge whether the entity corresponding to the query targets and the query conditions are the same. ① If the two corresponding entities are the same entity, the generation form of WHERE clause can not consider the entity association relationship. The generation results are as follows: WHERE entity. attribute name = |LIKE 'attribute value' [AND| OR entity. attribute name = | LIKE 'attribute value'] [ORDER BY attribute name]; ② If the two entities are different, the generation form of WHERE clause needs to consider the entity association relationship, and add the path generated by the association path algorithm to the query condition string.

(3) If the *GA* is not empty, initialize the grouping string, add 'GROUP BY', and then fill the *GA* field into the string. Finally, all strings are spliced into a complete SQL statement and returned.

The generated complete SQL query statement is submitted to the database management system of the application system, and the query result is returned to the user after the statement is processed.

4 Experiment and verification

4.1 Experimental data set and evaluation index

The data set in this paper is provided by the population Department of Shanghai Public Security Bureau. The data consists of two parts: common query questions of public security population and database structure. There are 100 common query questions, including 40 single table query questions, 40 multi-table query questions and 20 complex query questions. An example of query statement is shown in Tab.1. The database structure contains 1143 tables and table metadata, which contains the meta information of the table (for example, table size, field attributes, etc.). Note that due to data sensitivity, the data does not contain personal privacy related data. We will use ***precision*** as the metric to evaluate the effectiveness of NLIDB.

Table 1. Examples of query statement data

Single table query	Show me the delisted housing codes in Xuhui District.
	What is the ID number of the person named Zhang San?
	What is the total population of Huangpu District?
Multi-table query	Show me the work unit of the delisted house owner in Huangpu District.
	What is the domicile address of Ctrip's personnel?
	What is the provident fund for teachers of Donghua University?
Complex query	Show me the contact details of the courier recipient before April 2020.
	What is the name of the student who did not graduate?
	Show me the population of all Jiuting Streets in Songjiang District in descending order.

4.2 Experimental results and discussion

Only if the generated SQL statement is syntactically correct, and the query results through the SQL statement meet the user's query intention, we mark it a correct transformation. This paper will analyse and evaluate the effectiveness of NLIDB from multiple perspectives, including single-table query, multi-table query and complex query.

Precision: The ratio of the number of correctly retrieved samples to the total number of retrieved samples, that is, the ratio of correctly executed query statements to the total number of test set statements.

The results of different queries are shown in Fig.4. The precision rate of single table query, multi-table query and complex query is 83.3%, 68.8% and 51.9% respectively. The main reason for the failure of single table query is that the table where the query target is located is not found. The main reason for the failure of multi-table query is that the value condition query cannot correspond to the entity object and the generation error of association path. The main reason for the failure of complex query example sentence query is that there are deeply nested queries and the generation error of aggregation condition clauses.

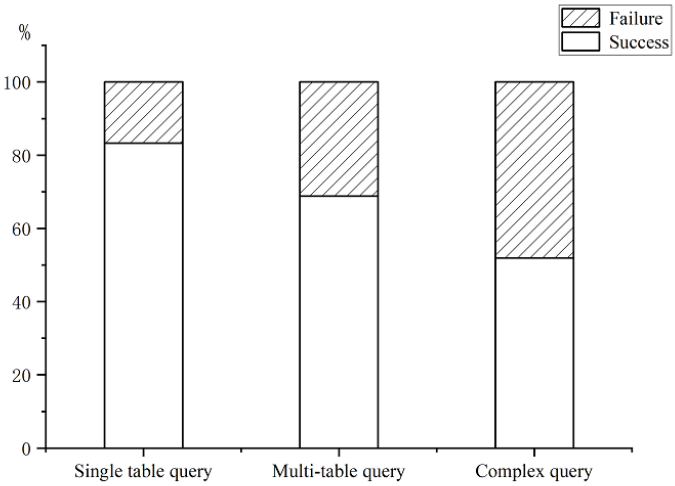

Fig. 4. Precision of different queries

The evaluation of the conversion effect of various types of query targets and query conditions is shown in Fig.5. The ***precision*** of explicit target and aggregation target is 79.4% and 64.1% respectively. There are several reasons that lead to the failure of natural language transformation into SQL: the standard words expressed by users' synonyms are not found, the query statements do not comply with grammar rules and restricted grammar, and there are compound concepts. In the query conditions, the precision rate of value-name condition is 78.4%, the precision rate of value condition is 70.3%, and the precision rate of aggregation condition is 56.5%. In addition, the usage frequency of value-name and value condition is higher than other conditions.

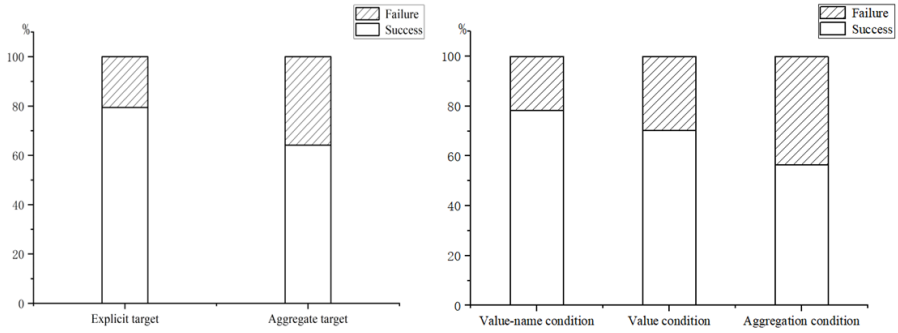

Fig. 5. Precision of different query targets and query conditions

5 Conclusion

In this paper, we propose an ontology-based NLIDB approach combining with public security population ontology and syntactic analysis. The ontology is first constructed by parsing and mapping the schema of the PSP-DB to integrate the public security domain knowledge effectively. After that, we propose an association path processing algorithm to handle multi-table connection path in SQL generation. The evaluation results show that the ontology-based NLIDB is effective and could fit the query scenario of public security population database well.

In the future, we will propose the algorithm to solve the problem of deep nesting and improve the accuracy of complex queries. In addition, the Chinese Word Segmentation can be optimized in combination with the deep learning of Named Entity Recognition (NER).

References

1. He P, Mao Y, Chakrabarti K, et al. X-SQL: reinforce schema representation with context [J]. arXiv, 2019.
2. Brunner U, Stockinger K. ValueNet: A Natural Language-to-SQL System that Learns from Database Information [J]. 2020.
3. Das A, Balabantaray R C. MyNLIDB: a natural language interface to database[C]//2019 International Conference on Information Technology (ICIT). IEEE, 2019: 234-238.
4. Affolter K, Stockinger K, Bernstein A. A Comparative Survey of Recent Natural Language Interfaces for Databases [J]. The VLDB Journal, 2019.
5. F. Li and H. V. Jagadish. Constructing an Interactive NaturalLanguage Interface for Relational Databases.PVLDB,8(1):73–84, 2014.
6. Saha D, Floratou A, Sankaranarayanan K, et al. ATHENA: An ontology-driven system for natural language querying over relational data stores[J].Proceedings of the VLDB Endowment, 2016, 9(12):1209-1220.
7. Jaydeep Sen, Chuan Lei, Abdul Quamar, et al. ATHENA++: natural language querying for complex nested SQL queries. Proc. VLDB Endow. 13, 12 (August 2020), 2747–2759.
8. Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. arXiv preprint arXiv:1709.00103, 2017.
9. Xu X, Liu C, Song D. SQLNet: Generating Structured Queries from Natural Language Without Reinforcement Learning [J]. 2017.
10. McGuinness D L, Van Harmelen F. OWL web ontology language overview [J]. W3C recommendation, 2004, 10(10): 2004.
11. Kou L, Markowsky G, Berman L. A fast algorithm for Steiner trees [J]. Acta informatica, 1981, 15(2): 141-145.