DuSQL: A Large-Scale and Pragmatic Chinese Text-to-SQL Dataset

Lijie Wang[1], **Ao Zhang**[1], **Kun Wu**[2], **Ke Sun**[1], **Zhenghua Li**[2],
Hua Wu[1], **Min Zhang**[2], **Haifeng Wang**[1]
1. Baidu Inc, Beijing, China
2. Institute of Artificial Intelligence, School of Computer Science and Technology,
Soochow University, Suzhou, China
{wanglijie,zhangao,sunke,wu_hua,wanghaifeng}@baidu.com
kwu@stu.suda.edu.cn; {zhli13,minzhang}@suda.edu.cn

Abstract

Due to the lack of labeled data, previous research on text-to-SQL parsing mainly focuses on English. Representative English datasets include ATIS, WikiSQL, Spider, etc. This paper presents DuSQL, a larges-scale and pragmatic Chinese dataset for the cross-domain text-to-SQL task, containing 200 databases, 813 tables, and 23,797 question/SQL pairs. Our new dataset has three major characteristics. First, by manually analyzing questions from several representative applications, we try to figure out the true distribution of SQL queries in real-life needs. Second, DuSQL contains a considerable proportion of SQL queries involving row or column calculations, motivated by our analysis on the SQL query distributions. Finally, we adopt an effective data construction framework via human-computer collaboration. The basic idea is automatically generating SQL queries based on the SQL grammar and constrained by the given database. This paper describes in detail the construction process and data statistics of DuSQL. Moreover, we present and compare performance of several open-source text-to-SQL parsers with minor modification to accommodate Chinese, including a simple yet effective extension to IRNet for handling calculation SQL queries.

1 Introduction

In the past few decades, a large amount of research has focused on searching answers from unstructured texts given natural questions, which is also known as the question answering (QA) task (Burke et al., 1997; Kwok et al., 2001; Allam and Haggag, 2012; Nguyen et al., 2016). However, a lot of high-quality knowledge or data are actually stored in databases in the real world. It is thus extremely useful to allow ordinary users to directly interact with databases via natural questions. To meet this need, researchers have proposed the text-to-SQL task with released English datasets for model training and evaluation, such as ATIS (Iyer et al., 2017), GeoQuery (Popescu et al., 2003), WikiSQL (Zhong et al., 2017), and Spider (Yu et al., 2018b).

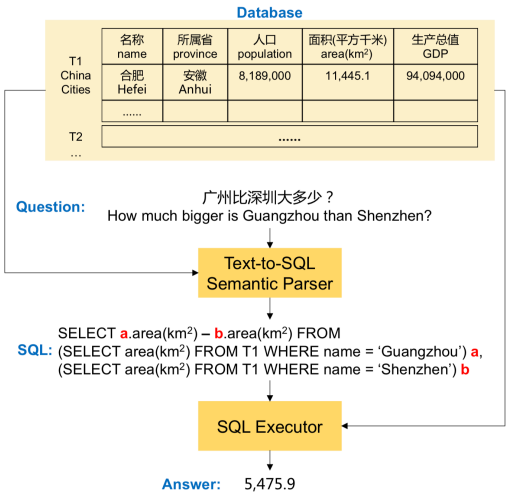

Figure 1: Illustration of the text-to-SQL task.

Formally, given a natural language (NL) question and a relational database, the text-to-SQL task aims to produce a legal and executable SQL query that leads directly to the correct answer, as depicted in Figure 1. A database is composed of multiple tables and denoted as DB $= \{T_1, T_2, ..., T_n\}$. A table is composed of multiple columns and denoted as $T_i = \{col_1, col_2, ..., col_m\}$. Tables are usually linked with each other by foreign keys.

The earliest datasets include ATIS (Iyer et al., 2017) , GeoQuery (Popescu et al., 2003), Restaurants (Tang and Mooney, 2001), Academic (Li and Jagadish, 2014), etc. Each dataset only has a single database containing a certain number of tables. All question/SQL pairs of train/dev/test sets are generated against the same database. Many interesting approaches are proposed to handle those datasets (Iyer et al., 2017; Yaghmazadeh et al., 2017; Finegan-Dollak et al., 2018).

However, real-world applications usually in-

Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, pages 6923–6935,
November 16–20, 2020.

volve more than one database, and require the model to be able to generalize to and handle unseen databases during evaluation. To accommodate this need, the WikiSQL dataset is then released by Zhong et al. (2017). It consists of 80,654 question/SQL pairs for 24,241 single-table databases. They propose a new data split setting to ensure that databases in train/dev/test do not overlap. However, they focus on very simple SQL queries containing one SELECT statement with one WHERE clause. In addition, Sun et al. (2020) released TableQA, a Chinese dataset similar to the WikiSQL dataset.

Yu et al. (2018b) released a more challenging Spider dataset, consisting of 10,181 question/SQL pairs against 200 multi-table databases. Compared with WikiSQL and TableQA, Spider is much more complex due to two reasons: 1) the need of selecting relevant tables; 2) many nested queries and advanced SQL clauses like GROUP BY and ORDER BY.

As far as we know, most existing datasets are constructed for English. Another issue is that they do not refer to the question distribution in real-world applications during data construction. Taking Spider as an example. Given a database, annotators are asked to write many SQL queries from scratch. The only requirement is that SQL queries have to cover a list of SQL clauses and nested queries. Meanwhile, the annotators write NL questions corresponding to SQL queries. In particular, all these datasets contain very few questions involving calculations between rows or columns, which we find are very common in real applications.

This paper presents DuSQL, a large-scale and pragmatic Chinese text-to-SQL dataset, containing 200 databases, 813 tables, and 23,797 question/SQL pairs. Specifically, our contributions are summarized as follows.

- In order to determine a more realistic distribution of SQL queries, we collect user questions from three representative database-oriented applications and perform manual analysis. In particular, we find that a considerable proportion of questions require row/column calculations, which are not included in existing datasets.

- We adopt an effective data construction framework via human-computer collaboration. The basic idea is automatically generating SQL queries based on the SQL grammar and constrained by the given database. For each SQL query, we first generate a pseudo question by traversing it in the execution order and then ask annotators to paraphrase it into a NL question.

- We conduct experiments on DuSQL using three open-source parsing models. In particular, we extend the state-of-the-art IRNet (Guo et al., 2019) model to accommodate the characteristics of DuSQL. Results and analysis show that DuSQL is a very challenging dataset. We will release our data at https://github.com/luge-ai/luge-ai/tree/master/semantic-parsing.

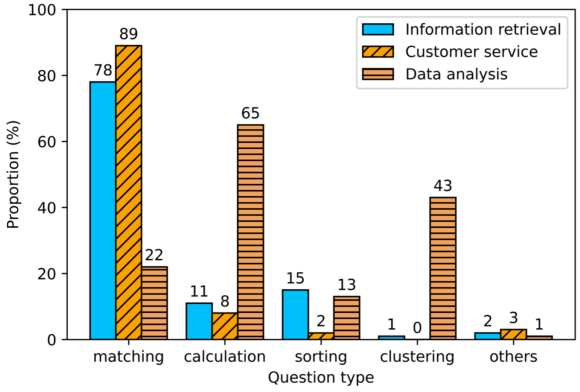

Figure 2: The SQL query distributions of the three applications. Please kindly note that a query may belong to multiple types.

2 SQL Query Distribution

As far as we know, existing text-to-SQL datasets mainly consider the complexity of SQL syntax when creating SQL queries. For example, WikiSQL has only simple SQL queries containing SELECT and WHERE clauses. Spider covers 15 SQL clauses including SELECT, WHERE, ORDER BY, GROUP BY, etc, and allows nested queries.

However, to build a pragmatic text-to-SQL system that allows ordinary users to directly interact with databases via NL questions, it is very important to know the SQL query distribution in real-world applications, from the aspect of user need rather than SQL syntax. Our analysis shows that Spider mainly covers three types of SQL queries, i.e., matching, sorting, and clustering, whereas WikiSQL only has matching queries. Neither of them contains the calculation type, which we find composes a large portion of questions in certain real-world applications.

To find out the SQL query distribution in real-life applications, we consider the following three

representative types of database-oriented applications, and conduct manual analysis against user questions. We ask annotators to divide user questions into five categories (see Appendix B for details), i.e., *matching*, *sorting*, *clustering*, *calculation*, and *others*.

Information retrieval applications. We use Baidu, the Chinese search engine, as a typical information retrieval application. Nowadays, search engines are still the most important way for web users to acquire answers. Thanks to the progress in knowledge graph research, search engines can return structured tables or even direct answers from infobox websites such as Wikipedia and Baidu Encyclopedia. From one-day Baidu search logs, we randomly select 1,000 questions for which one of returned top-10 relevant web sites is from infobox websites. Then, we manually classify each question into the above five types.

Customer service robots. Big companies build AI robots to answer questions of customers, which usually require the access to industrial databases. We provide a free trial API[1] to create customer service robots for developers. With the permission of the developers, we randomly select 1,500 questions and corresponding databases from their created robots. These questions cover multiple domains such as banks, airlines, and communication carriers, etc.

Data analysis robots. Every day, innumerous tables are generated, such as financial statements, business orders, etc. To perform data analysis over such data, companies hire professionals to write SQL queries. Obviously, it is extremely useful to build robots that allow financial experts to directly perform data analysis using NL questions. We collect 500 questions from our data analysis robot.

Figure 2 shows the query distributions of the three applications. It is obvious that calculation questions occupy a considerable proportion in all three applications. For customer service robots, users mainly try to search information, and therefore most questions belong to the matching type. Yet, 8% questions require calculation SQL queries to be answered. For data analysis robots, calculation questions dominate the distribution, since users try to figure out useful clues behind the data.

To gain more insights, we further divide calculation questions into three subtypes according to the SQL syntax, i.e., row calculation, column calculation, and calculation with a constant. Figure 3 shows some examples.

[1]The API is publicly available at https://ai.baidu.com/unit/v2#/innovationtec/home.

Row Calculation

How much bigger is Guangzhou than Shenzhen?

SELECT a.area(km^2) - b.area(km^2) FROM
(SELECT area(km^2) FROM T1 WHERE name = 'Guangzhou') a,
(SELECT area(km^2) FROM T1 WHERE name = 'Shenzhen') b

Column Calculation

What is the population density of Hefei?

SELECT population / area(km^2) FROM T1 WHERE name = 'Hefei'

Calculation with a Constant

How old is Jenny?

SELECT curdate - birthday FROM student WHERE name = 'Jenny'

How far is Beijing's population from 23 million?

SELECT 23000000 - population FROM T1 WHERE name = 'Beijing'

Figure 3: Examples in the calculation type, including questions and SQL queries. The first example of *calculation with a constant* is based on a database that has a "student" table with the schema of {*name, birthday, height, age*}. Other examples are based on the database in Figure 1.

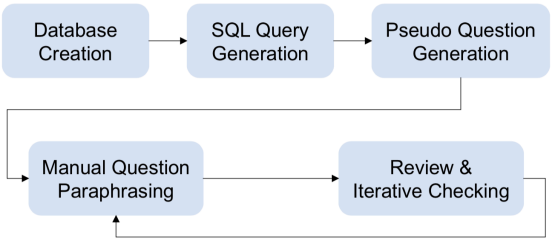

Figure 4: The construction workflow of DuSQL.

3 Corpus Construction

Building a large-scale text-to-SQL dataset with multi-table databases is extremely challenging. First, though there are a large amount of independent tables on the Internet, connections among the tables are usually unavailable. Therefore, great efforts are needed to create multi-table databases. Second, it is usually difficult to obtain NL questions against certain databases. Third, given a question and the corresponding database, we need proficient annotators to write a SQL query for the question who understand both the database schema and the SQL syntax.

Different from previous works, which usually rely on human to create both NL questions and SQL queries (Yu et al., 2018b), we build our dataset via a human-computer collaboration way,

as illustrated in Figure 4. The key idea is to automatically generate SQL queries paired with pseudo questions given a database. Then pseudo questions are paraphrased to NL questions by humans. Finally, to guarantee data quality, low-confidence SQL queries and NL questions detected according to their overlapping and similarity metrics, and are further checked by humans.

3.1 Database Creation

Most of mature databases used in industry are not publicly available. So we collect our databases mainly from the Internet. However, databases available on the Internet are in the form of independent tables, which need to be linked with other tables. We create databases in three steps: table acquisition, table merging, and foreign key creation.

We collect websites to crawl tables, ensuring that they cover multiple domains. As the largest Chinese encyclopedia, Baidu Baike contains more than 17 million entries across more than 200 domains. We start with all the entries in Baike as the initial sites, and extend the collection based on the reference sites in each entry page. We keep sites where tables are crawled. The final collection contains entries of Baike, annual report websites[2], vertical domain websites[3], and other websites such as community forums[4]. Table 1 shows the data distribution regarding database sources.

To make a domain correspond to a database, we merge tables with the same schema to a new table with a new schema, e.g., tables about China cities with the schema of $\{population, area, ...\}$ are merged to a new table with the schema of $\{termid, name, population, area, ...\}$, where $termid$ is randomly generated as primary key and $name$ is the name of the city. Meanwhile, we add a type for each column according to the form of its value, where the column type consists of text, number and date.

We create foreign keys between two tables via entity linking, e.g., a table named "Livable cities in 2019" with the schema of {*city_name, ranker, ...*} joins to a table named "China cities" with the schema of {*term_id, name, area, ...*} through the links of entities in "city_name" and "name". According to foreign keys, all tables are split into separate graphs, each of which consists of several joined tables. We choose 200 graphs to create databases, and manually check and correct foreign keys for each database.

Source	Proportion
Baike	40.3
Vertical domain websites	31.3
Annual report	23.4
Others	5.0

Table 1: The distribution of database sources.

Overall, we create 200 databases with 813 tables, covering about 70% of Baike entries from more than 160 domains such as movies, actors, cities, animals, foods, etc. Since some tables are sensitive, we use the column header of each table, and populate it with randomly selected values from the original table.

3.2 Automatic Generation of SQL Queries

Given a database, we want to generate as many common SQL queries as possible. Both manually writing SQL queries and quality-checking take a significant amount of time. Obviously, SQL queries can be automatically generated from the grammar. We utilize production rules from the grammar to automatically generate SQL queries, instead of asking annotators to write them. According to the difficulty[5] and semantic correctness of a SQL query, we prune the rule paths in the generation. Then, we sample the generated SQL queries according to the distribution in Figure 2 and carry out the follow-up work based on them.

As illustrated in Figure 5, the SQL query can be represented as a tree using the rule sequence of {*SQLs = SQL, SQL = Select Where, Select = SELECT A, Where = WHERE Conditions, ...*}, all of which are production rules of the grammar. Guided by the SQL query distributions in real applications, we design production rules to ensure that all common SQL queries can be generated, e.g., the rule of {*C = table.column mathop table.column*} allows calculations between columns or rows. By exercising every rule of the grammar, we can generate SQL queries covering patterns of different complexity.

We consider two aspects in the automatic SQL generation: the difficulty and semantic correct-

[2]QuestMobile, 199it, tianyancha, etc.

[3]State Statistical Bureau, China Industrial Information Network, Shopping websites, Booking websites, etc.

[4]Baidu Tieba, Newsmth, Hupu, etc.

[5]We observe that very complex queries are rare in search logs. Since our SQL queries are automatically generated, without complexity control, the proportion of complex queries would dominate the space, thus deviating from the real query distribution.

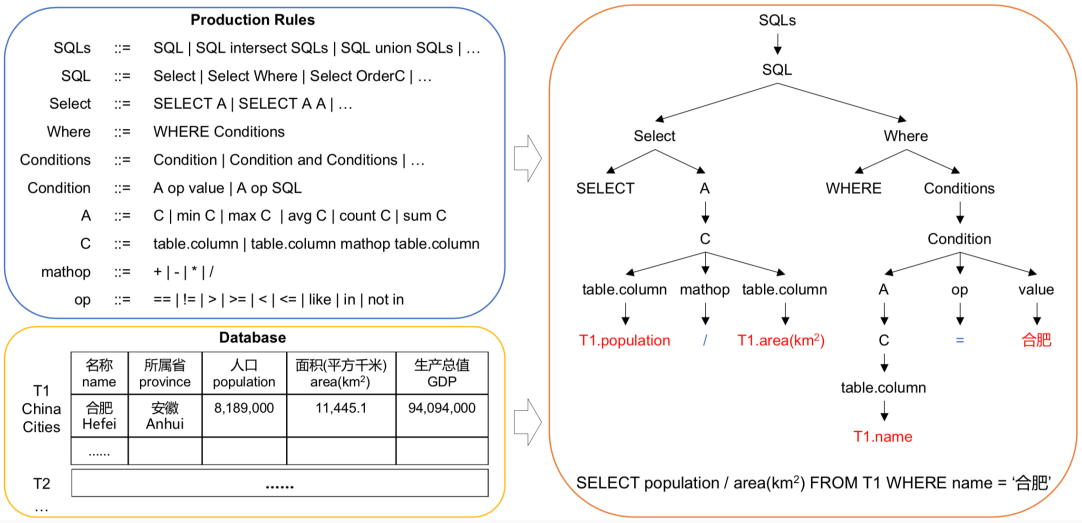

Figure 5: An example of SQL query generation from the grammar. We show a part of production rules (all rules are shown in Appendix A). The leaf nodes in red are from the database.

ness of a SQL query. To control the difficulty of the generated queries, we make some restrictions based on our analysis on real-life questions: first, a SQL query contains only one nested query; second, there are no more than three conditions in a where clause and no more than four answers in a select statement; third, a SQL query has at most one math operation; forth, most text values are from databases[6]. To ensure the semantics correctness of the generated query, we abide by preconditions of each clause and expression in the generation, e.g., the expression of {A > SQL} requires that the nested SQL returns a number value. The full list of preconditions is shown in Appendix C.

Under these requirements, we generate a large amount of candidate SQL queries against 200 databases. Among them, only a tiny proportion of SQL queries are of the calculation type, since only few columns support calculation operations. We keep all queries in the calculation type, randomly select ones with sorting and clustering types of the same size, and select ones with the matching type[7] of three times the size. We make sure that these selected queries are spread across all 200 databases. Then these queries are used as input for the follow-up work.

[6]The text values in a SQL query are from the database to reduce the difficulty of SQL prediction. We plan to remove this restriction in the next release version of DuSQL.

[7]Including combinations of matching type and other types, e.g., the SQL query of {SELECT ... WHERE ... ORDER BY ... } represents the combination of matching and sorting types.

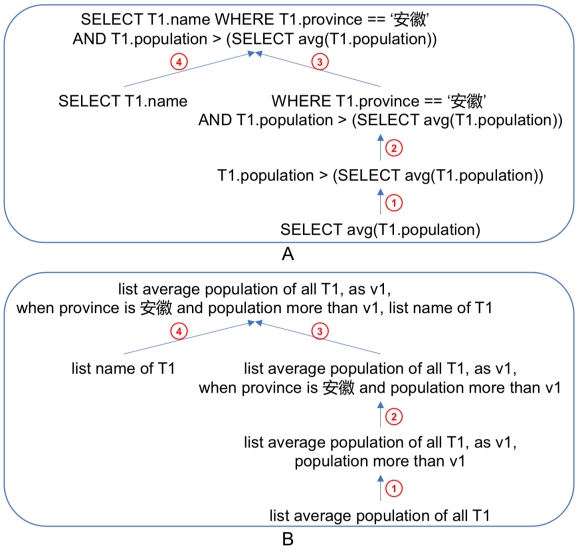

Figure 6: An example of the pseudo question generation according to the execution order of the SQL query. The numbers in circles represent the order of execution.

3.3 Semi-automatic Generation of Questions

For each SQL query, we automatically generate a pseudo question to explain it. Then pseudo questions are shown to annotators who can understand them and paraphrase them to NL questions without looking at databases and SQL queries.

We generate a pseudo question for a SQL query according to its execution order. As shown in Figure 6, the entire pseudo question of the SQL query consists of pseudo descriptions of all clauses according to their execution orders. The pseudo description of a clause consists of pseudo descriptions of all its components. We give a description

for each component, e.g., *list* for *SELECT*, *average* for the aggregator of *avg*. Appendix D shows the descriptions for all components. To ensure that the pseudo question is clear and reflects the meaning of the SQL query, intermediate variables are introduced to express sub-SQL queries, e.g., "v1" in the example of Figure 6 represents the result of the nested query and is used as a value in other expressions.

We ask two annotators[8] to reformulate pseudo questions into NL questions[9], and filter two kinds of questions: 1) incomprehensible ones which are semantically unclear; 2) unnatural ones which are not the focus of humans[10]. During the process of paraphrasing, 6.7% of question/SQL pairs are filtered, among which 76.5% are complex queries. Then we ask other annotators to check the correctness of reformulated questions, and find 8% of questions are inaccurate.

3.4 Review and Checking

To guarantee data quality, we automatically detect low-quality question/SQL pairs according to the following evaluation metrics.

- **Overlap**. To ensure the naturalness of our questions, we calculate the overlap between the pseudo question and the corresponding NL question. The question with an overlap higher than 0.6 is considered to be of low quality.
- **Similarity**. To ensure that the question contains enough information for the SQL query, we train a similarity model based on question/SQL pairs. The question with a similarity score less than 0.8 is considered to be of low quality.

In the first round, about 18% of question/SQL pairs are of low quality. We ask annotators to check these pairs and correct the error pairs. This process iterates through the collaboration of human and computer until the above metrics no longer changing. It iterates twice in the construction of DuSQL.

3.5 Dataset Statistics

We summarize the statistics of DuSQL and other cross-domain datasets in Table 2, and give some examples in Figure 7. DuSQL contains enough question/SQL pairs for all common types. WikiSQL and TableQA are simple datasets, only containing matching questions. Spider and CSpider (Min et al., 2019) mainly cover matching, sorting, clustering and their combinations. There are very few questions in the calculation type, and all of them only need column calculations. Spider does not focus on questions that require the common knowledge and math operation. According to our analysis in Figure 2, the calculation type is very common, accounting for 8% to 65% in different applications. DuSQL, a pragmatic industry-oriented dataset, conforms to the distribution of SQL queries in real applications. Meanwhile, DuSQL is larger, twice the size of other complex datasets. DuSQL contains 200 databases, covering about 70% of entries in Baike and more than 160 domains, e.g., cities, singers, movies, animals, etc. We provide content for each database. All the values of a SQL query can be found in the database, except for numeric values. All table and column names in the database are clear and self-contained. In addition, we provide English schema for each database, including table names and column headers.

哪些城市首套房首付比例不少于20%？
Which cities have a down payment ratio no less than 20% for the first home?

SELECT 名称 FROM 城市 WHERE 首套房首付比例 >= 0.2
SELECT} name FROM city WHERE down_payment_ratio_first >= 0.2

占地最大的3个图书馆位于哪个城市？具体哪里？
Which cities are the 3 largest libraries located in? Where is the location?

SELECT T2.名称, T1.地址 FROM 图书馆 AS T1 JOIN 城市 AS T2
ON T1.所属城市id = T2.词条id ORDER BY T1.面积 DESC LIMIT 3
SELECT T2.name, T1.address FROM library AS T1 JOIN city AS T2
ON T1.city_id = T2.termid ORDER BY T1.area DESC LIMIT 3

哪些学校本科就业学生数等于所有高校本科就业人数的平均值？
For which university, the number of graduates for direct employment is equal to the average of all universities?

SELECT 名称 FROM 高校 WHERE 本科毕业人数 - 继续深造人数
= (SELECT avg (本科毕业人数 - 继续深造人数) FROM 高校)
SELECT name FROM college WHERE graduate_number - further_number
= (SELECT avg (graduate_number - further_number) FROM college)

既有相关电视剧又有相关电影的皇帝有哪些？
List the emperors who appear both in TV dramas and movies?

(SELECT 帝王id FROM 电视剧) INTERSECT (SELECT 帝王id FROM 电影)
(SELECT emperor_id FROM tv) INTERSECT (SELECT emperor_id FROM movie)

Figure 7: SQL query examples.

4 Benchmark Approaches

All existing text-to-SQL works focus on English datasets. Considering that DuSQL is the most similar with Spider, we choose the following three representative publicly available parsers as our benchmark approaches, to understand the performance of existing approaches on our new Chinese dataset.

[8]They are full-time employees and familiar with SQL language. Meanwhile, they have lots of experience in annotating QA data.

[9]Some values in SQL queries are rewritten as synonyms.

[10]E.g., "*When province is Sichuan, list the total rank of these cities.*" for the SQL query {SELECT sum(rank) From T2 WHERE province = 'Sichuan'} is considered as an unnatural question, as *the total rank* would not be asked by humans.

Dataset	Size	DB	Table/DB	Matching	Sorting	Clustering	Calculation			Others
							Column	Row	Constant	
WikiSQL	80,654	26,251	1	80,654	0	0	0	0	0	0
TableQA	49,974	5,291	1	49,974	0	0	0	0	0	0
Spider	9,693	200	5.1	6,450	863	1,059	13	0	0	1,308
CSpider	9,691	166	5.3	6,448	862	1,048	13	0	0	1,318
Ours	23,797	200	4.1	6,440	2,276	3,768	1,760	1,385	1,097	7,071

Table 2: Statistics and comparisons of all existing cross-domain text-to-SQL datasets. The statistics of Spider are based on published data, only containing train and development sets. *Others* consists of combinations between matching, sorting and clustering types.

We also extend the state-of-the-art IRNet model of Guo et al. (2019) to accommodate the two characteristics of our data, i.e., calculation questions and the need of value prediction.

Seq2Seq+Copying (Zhong et al., 2017) incorporates the database schemas into the model input and uses a copying mechanism in the decoder.

SyntaxSQLNet (Yu et al., 2018a) proposes a SQL syntax tree-based network to generate SQL structures, and uses generation path history and table-aware column attention in the decoder.

IRNet (Guo et al., 2019) designs an intermediate representation called SemQL for encoding higher-level abstraction structures than SQL, and then uses a grammar-based decoder (Yin and Neubig, 2017) to synthesize a SemQL query. At present, IRNet reports the state-of-the-art results on Spider dataset.

Both SyntaxSQLNet and IRNet utilize a grammar to guide SQL generation and conduct experiments on Spider dataset. However, neither of their grammars can handle calculation questions. Another major difference between our dataset and Spider is that our evaluation metric (see Section §5) also considers value prediction, since values in a SQL query are from the corresponding question or database both of which are available inputs to the model. Please refer to our discussion in Section §3 for details. Due to the characteristics of our dataset, all the three models perform poorly on DuSQL. Therefore, we extend the IRNet model to accommodate DuSQL as follows.

Firstly, we extend the grammar of SemQL to accommodate the two characteristics of our dataset, as shown in Figure 8. The production rules in bold are added to parse calculation questions. Other production rules are modified based on original rules to support value prediction (Due to space limitation, we attach the full list of extended grammar in Appendix F.). Then we use all the n-grams of length 1-6 in the question to match database cells

$$
\begin{aligned}
\boldsymbol{Z} &::= + R\,R \mid - R\,R \mid \times R\,R \mid \div R\,R \\
\textit{Filter} &::= = A\,V \mid != A\,V \mid > A\,V \mid < A\,V \\
&\quad \mid >= A\,V \mid <= A\,V \mid \textit{like}\ A\,V \\
\textit{Superlative} &::= \textit{des}\ A\,V \mid \textit{asc}\ A\,V \\
\boldsymbol{A} &::= \textit{max MathA} \mid \textit{min MathA} \mid \textit{count MathA} \\
&\quad \mid \textit{sum MathA} \mid \textit{avg MathA} \mid \textit{none MathA} \\
\boldsymbol{MathA} &::= + A\,A \mid - A\,A \mid \times A\,A \mid \div A\,A \\
\boldsymbol{V} &::= \textit{value}
\end{aligned}
$$

Figure 8: The extended grammar for SemQL.

or number/date to determine candidate values for the predicated SQL query. The values are used in the same way as the columns and tables in the IRNet model.

5 Experiments

Data Settings Following WikiSQL, we split our dataset into train/dev/test in a way so that databases are non-overlapping among the three subsets. In other words, all question/SQL pairs for the same database are in the same subset. This is also referred to as cross-domain parsing problem, since some database schemes in dev/test do not appear in train. At last, 200 databases are split into 160/17/23, and 23,979 question/SQL pairs are split into 18,602/2,039/3,156.

Evaluation Metrics Evaluation metrics for the text-to-SQL task include component matching, exact matching, and execution accuracy. Component matching (Yu et al., 2018b) uses F1 score to evaluate the performance of the model on each clause. Exact matching, namely the percentage of questions whose predicted SQL query is equivalent to the gold SQL query, is widely used in text-to-SQL tasks. Execution accuracy, namely the percentage of questions whose predicted SQL query obtains the correct answer, assumes that each SQL query has an answer.

We use exact matching as the main metric, and follow Xu et al. (2017) and Yu et al. (2018b) to

Methods	Calculation				Matching	Sorting	Clustering	Others
	Column	Row	Constant	All				
IRNet	0	0	0	0	25.0	32.8	34.2	8.7
IRNetExt	22.0	34.3	37.9	29.7	52.1	68.7	60.8	52.5

Table 3: Performances of different SQL query types.

Methods	w/o values		w/ values	
	Dev	Test	Dev	Test
Seq2SeqCopying	6.6	3.9	2.6	1.9
SyntaxSQLNet	14.6	8.6	7.1	5.2
IRNet	38.4	34.2	18.4	15.4
IRNetExt	59.8	54.3	56.2	50.1
w/o calculation	50.2	48.2	46.6	45.6
w/o value	50.1	43.5	19.4	17.9

Table 4: Performance of the benchmark approaches.

handle the "ordering issue". Finally, we give the model performance with (w) and without (w/o) value evaluation.

Main results. Table 4 shows performance of the benchmark approaches. The performance of Seq2SeqCopying is the lowest. It uses the copying mechanism to reduce errors posed by out-of-domain words in the databases of test set. But it predicts lots of invalid SQL queries with grammatical errors, since its decoder does not consider SQL structures at all.

SyntaxSQLNet and IRNet outperform Seq2SeqCopying by utilizing a grammar from SQL structures to guide SQL generation. In particular, IRNet utilize SemQL as an abstraction representation of SQL queries. However, neither of the two vanilla models handles calculation questions and value directions properly. The basic IRNet achieves only 34.2/15.4 accuracy on the test set w/o and w/ value evaluation.

We can see that by simply extending IRNet to parse calculation questions and predict values, the IRNetExt model achieves much higher accuracy (54.3/50.1).

Ablation study. We perform ablation study to gain more insights on the contribution of our extensions. As shown in table 4, the accuracy on test set drops 4.5 by excluding production rules from the grammar of SemQL. The accuracy of calculation type is 0, which composes 20.7% of the questions in the test set. After excluding the prediction of values, the test performance drops significantly for two reasons. First, there are a large number of questions that contain values, accounting for about 75% in the dev set and 70% in the test set. Second, the generation of where clauses can be improved by leveraging the column-cell relationship.

Analysis. Table 3 shows performance of different SQL query types. Firstly, the grammar extension is effective, the accuracy of all types is significantly improved. Second, the accuracy of calculation type is lower than that of other types, as many calculation questions require incorporating common knowledge, e.g., *age* = *dateOfDeath* - *dateOfBirth*. How to represent and incorporate such knowledge into the model is very challenging. Third, questions requiring common knowledge perform poorly, as they need understanding rather than matching, such as the matching issue of "the oldest" and "age".

6 Related Work

Semantic parsing. Semantic parsing aims to map NL utterances into semantic representations, such as logical forms (Liang, 2013), SQL queries (Tang and Mooney, 2001), Python code (Ling et al., 2016), etc. In order to facilitate model training and evaluation, researchers release a variety of datasets. ATIS and GeoQuery are two popular early datasets originally in logical forms, and are converted into SQL queries (Iyer et al., 2017; Popescu et al., 2003). As two recently released datasets, WikiSQL (Zhong et al., 2017) and Spider (Yu et al., 2018b) have attracted extensive research attention. It is also noteworthy that Min et al. (2019) propose the CSpider dataset by translating English questions of Spider into Chinese.

Data construction methods. As discussed in Section §3, creating a large-scale semantic parsing dataset is extremely challenging. To construct Spider, Yu et al. (2018b) ask annotators to write both questions and SQL queries given a database. Both Iyer et al. (2017) and Herzig and Berant (2019) assume that the database and questions are given and try to reduce the effort of creating semantic representations. Our data construction is most closely related to Overnight (Wang et al., 2015), who proposes to automatically generate logical forms based on a hand-crafted grammar and ask annotators to paraphrase pseudo questions into NL ques-

tions. Overnight focuses on logic form (LF) based semantic representation, while our work on SQL representation. The differences are two-fold. First, databases of Overnight are much simpler, composed of a set of entity-property-entity triples. Second, LF operations of Overnight are much simpler, consisting of only matching and aggregation operations, such as count, min, max. Our dataset is more complex and thus imposes more challenges on the data construction.

Text-to-SQL parsing approaches. Seq2Seq models achieve the state-of-the-art results on single-database datasets such as ATIS and GeoQuery (Dong and Lapata, 2016). With the release of WikiSQL dataset, researchers make efforts to handle unseen databases by using database schema as inputs. Two mainstream approaches are the Seq2Seq model with copy mechanism (Sun et al., 2018) and the Seq2Set model (Xu et al., 2017). With BERT representations (Devlin et al., 2019), the execution accuracy exceeds 90% (He et al., 2019; Guo and Gao, 2019).

For the more challenging Spider dataset with multi-table databases, Guo et al. (2019) introduces an intermediate representation (SemQL) for SQL queries, and uses a grammar-based decoder to generate SemQL, reporting state-of-the-art performance. Bogin et al. (2019) proposes to encode the database schema with graph neural network. Recently, Wang et al. (2019) proposes RATSQL to use relation-aware self-attention to better encode the question and database schema simultaneously.

7 Conclusion

We present the first large-scale and pragmatic Chinese dataset for cross-domain text-to-SQL parsing. Based on the analysis on questions from real-world applications, our dataset contains a considerable proportion of questions that require row/column calculations. We extend the state-of-the-art IR-Net model on Spider to accommodate DuSQL, and obtain substantial performance boost. Yet, there is still a large room for improvement, especially on calculation questions which usually require incorporation of common-sense knowledege into the model. For future work, we will continually improve the scale and quality of our dataset, to facilitate future research and to meet the need of database-oriented applications.

Acknowledgments

We thank three anonymous reviewers for their helpful feedback and discussion on this work. Zhenghua Li and Min Zhang were supported by National Natural Science Foundation of China (Grant No. 61525205, 61876116), and a Project Funded by the Priority Academic Program Development (PAPD) of Jiangsu Higher Education Institutions.

References

Ali Mohamed Nabil Allam and Mohamed Hassan Haggag. 2012. The question answering systems: A survey. *International Journal of Research and Reviews in Information Sciences (IJRRIS)*, 2(3).

Ben Bogin, Jonathan Berant, and Matt Gardner. 2019. Representing schema structure with graph neural networks for text-to-sql parsing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4560–4565.

Robin D Burke, Kristian J Hammond, Vladimir Kulyukin, Steven L Lytinen, Noriko Tomuro, and Scott Schoenberg. 1997. Question answering from frequently asked question files: Experiences with the faq finder system. *AI magazine*, 18(2):57–57.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 33–43.

Catherine Finegan-Dollak, Jonathan K Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving text-to-sql evaluation methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 351–360.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards complex text-to-sql in cross-domain database with intermediate representation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4524–4535.

Tong Guo and Huilin Gao. 2019. Content enhanced bert-based text-to-sql generation. *arXiv preprint arXiv:1910.07179*.

Pengcheng He, Yi Mao, Kaushik Chakrabarti, and Weizhu Chen. 2019. X-sql: reinforce schema representation with context. *arXiv preprint arXiv:1908.08113*.

Jonathan Herzig and Jonathan Berant. 2019. Dont paraphrase, detect! rapid and effective data collection for semantic parsing. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3801–3811.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 963–973.

Cody CT Kwok, Oren Etzioni, and Daniel S Weld. 2001. Scaling question answering to the web. In *Proceedings of the 10th international conference on World Wide Web*, pages 150–161.

Fei Li and HV Jagadish. 2014. Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment*, 8(1):73–84.

Percy Liang. 2013. Lambda dependency-based compositional semantics. *arXiv preprint arXiv:1309.4408*.

Wang Ling, Phil Blunsom, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiskỳ, Fumin Wang, and Andrew Senior. 2016. Latent predictor networks for code generation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 599–609.

Qingkai Min, Yuefeng Shi, and Yue Zhang. 2019. A pilot study for chinese sql semantic parsing. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3643–3649.

Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. Ms marco: a human-generated machine reading comprehension dataset.

Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. 2003. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th international conference on Intelligent user interfaces*, pages 149–157.

Ningyuan Sun, Xuefeng Yang, and Yunfeng Liu. 2020. Tableqa: a large-scale chinese text-to-sql dataset for table-aware sql generation. *arXiv preprint arXiv:2006.06434*.

Yibo Sun, Duyu Tang, Nan Duan, Jianshu Ji, Guihong Cao, Xiaocheng Feng, Bing Qin, Ting Liu, and Ming Zhou. 2018. Semantic parsing with syntax-and table-aware sql generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 361–372.

Lappoon R Tang and Raymond J Mooney. 2001. Using multiple clause constructors in inductive logic programming for semantic parsing. In *European Conference on Machine Learning*, pages 466–477. Springer.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2019. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. *arXiv preprint arXiv:1911.04942*.

Yushi Wang, Jonathan Berant, and Percy Liang. 2015. Building a semantic parser overnight. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1332–1342.

Xiaojun Xu, Chang Liu, and Dawn Song. 2017. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436*.

Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. 2017. Sqlizer: query synthesis from natural language. *Proceedings of the ACM on Programming Languages*, 1(OOPSLA):1–26.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 440–450.

Tao Yu, Michihiro Yasunaga, Kai Yang, Rui Zhang, Dongxu Wang, Zifan Li, and Dragomir Radev. 2018a. Syntaxsqlnet: Syntax tree networks for complex and cross-domain text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1653–1663.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018b. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.

SQLs	::=	SQL intersect SQLs | SQL union SQLs
		| SQL except SQLs | SQL
SQL	::=	Select | Select Where
		| Select GroupC | Select Where GroupC
		| Select OrderC | Select Where OrderC
		| Select from SQL, SQL
Select	::=	select A | select A A
		| select A A A | select A A A A
Where	::=	where Conditions
GroupC	::=	group by C
		| group by C having Conditions
		| group by C OrderC
OrderC	::=	order by C Dir | order by C Dir limit value
		| order A Dir limit value
Dir	::=	asc / desc
Conditions	::=	Condition | Condition and Conditions
		| Condition or Conditions
Condition	::=	A op value | A op SQL
A	::=	min C | max C | count C | sum C | avg C | C
C	::=	table.column
		| table.column1 mathop table.column2
		| table1.column mathop table2.column
mathop	::=	+ | - | * | /
op	::=	= | != | > | >= | < | <= | like | in | not in

Figure 9: The production rules for SQL generation.

A The Grammar for SQL Generation

Figure 9 shows production rules used for SQL generation. All kinds of SQL queries can be generated by exercising each rule, e.g., the rule of {*Condition = A op SQL*} for nested query generation, the rule of {*C = table.column1 mathop table.column2*} and {*C = table1.column mathop table2.column*}for calculation query generation.

B Query Type Definition

Question classification is mostly based on the operations used in corresponding SQL queries. Matching means the answer can be directly obtained from the database. Sorting means we need to sort the returned results or only return top-k results. Clustering means we have to perform aggregations (count, min/max, etc.) on each cluster. Calculation means we need to calculate between columns or rows to get the answer. Other usually corresponds to questions requiring reasoning or subjective questions, e.g., "*Is Beijing bigger than Shanghai?*", and "*Is the ticket expensive?*". Figure 10 shows some examples for types in Figure 2, except for the calculation type (shown in Figure 3) and other type which do not have corresponding SQL queries.

Matching

List cities with a population less than 10 million.

SELECT name FROM T1 WHERE population < 10000000

Sorting

Give the top 5 cities with the largest population.

SELECT name FROM T1 ORDER BY population DESC LIMIT 5

Clustering

Give the total population of each province.

SELECT province, sum(population) FROM T1 GROUP BY province

Figure 10: Examples of types in Figure 2. All of them are based on the database in Figure 1.

C Preconditions in SQL Generation

To ensure the semantic correctness of the generated SQL query, we define the preconditions for each production rule, and abide by these preconditions in the SQL query generation.

- For the generation of SQL query with multiple SQLs, e.g., {*SQLs ::=SQL union SQLs*}: the columns in the select clause of the previous SQL match the columns in the select clause of the subsequent SQL, i.e., the columns of the two select clauses are the same or connected by foreign keys.

- For the rule of generating *GroupC*: the *C* is generated from the rule of {*C ::= table.column*}, where the column can perform the clustering operation, that is to say, the table can be divided into several sub-tables according to the values of this column.

- For the rule of {*Condition ::= A op value*}: *op* ∈ {*<, <=, >, >=, =, !=, like*}. If *op* ∈ {*<, <=, >, >=*}, *A* and *value* must be in the type of number or date. If *op* is *like*, *A* must be in text type.

- For the rule of {*Condition ::= A op SQL*}: *op* ∈ {*<, <=, >, >=, =, !=, in, not in*}. If *op* ∈ {*<, <=, >, >=, =, !=*}, *A* and *SQL* must be in the type of number, and {*>= min, <= max*} are invalid. If *op* ∈ {*in, not in*}, *SQL* must return a set.

- For the rule of generating *A*: {*avg C* | *sum C*} require the *C* is in number type, {*min C* | *max C*} require the *C* is in number or date type, and {*count C*} requires the *C* is in text type.

Dataset	Size	DB	Table/DB	Order	Group	Having	Nest	Calculation		
								Column	Row	Constant
WikiSQL	80,654	26,251	1	0	0	0	0	0	0	0
TableQA	49,974	5,291	1	0	0	0	0	0	0	0
Spider	10,181	200	5.1	1,335	1,491	388	844	13	0	0
CSpider	9,691	166	5.3	1,052	1,123	505	913	13	0	0
Ours	23,797	200	4.1	4,959	3,029	3,432	2,208	1,760	1,385	1,097

Table 5: Comparisons of cross-domain text-to-SQL datasets. The statistics of Spider are from Yu et al. (2018b). The statistics of CSpider are based on the released data.

- For the rule of {*C ::= t1.column mathop t2.column*}: the two columns are of the same type, either number or date. Then we have to make sure that the columns are comparable based on rules built by search log analysis.
- For the rule of {*C ::= t1.column1 mathop t1.column2*}: the numerical units of these two columns can perform corresponding mathematical operations, e.g., *CNY/per* $\times$ *person* = *CNY*.

D Descriptions of SQL Components

We provide a description for each basic component, as follows:

- The descriptions for aggregators of {*min, max, count, sum, avg*} are {*minimum, maximum, the number of, total, average*}.
- The descriptions for operators of {=, *!=*, >, >=, <, <=, *like, in, not in*} are based on the column type. The descriptions for {=, *!=*, *like, in, not in*} with the text type are {*is, is not, contain, in, not in*}, descriptions for {=, *!=*, >, >=, <, <=} with the number type are {*is equal to, is not equal to, more than, no less than, less than, no more than*}, and descriptions for {=, *!=*, >, >=, <, <=} with the date type are {*in, not in, after, in or after, before, in or before*}.
- The descriptions for math operators of {+, -, *, /} are {*sum, difference, product, times*}.
- The descriptions for the condition relations {*and, or*} are {*and, or*}.
- The descriptions for {*asc, desc*} are {*in the ascending, in the descending*}.
- The descriptions for columns, tables, and values are equal to themselves.

Meanwhile, we provide the description for each production rule, as shown in Figure 12.

$$
\begin{aligned}
Z &::= \textit{intersect R R} \mid \textit{union R R} \mid \textit{except R R} \mid R \\
&\quad \mid + \; R \; R \mid - \; R \; R \mid \times \; R \; R \mid \div \; R \; R \\
R &::= \textit{Select} \mid \textit{Select Filter} \\
&\quad \mid \textit{Select Order} \mid \textit{Select Order Filter} \\
&\quad \mid \textit{Select Superlative} \mid \textit{Select Superlative Filter} \\
\textit{Select} &::= A \mid A \; A \mid A \; A \; A \mid \textit{A... A} \\
\textit{Filter} &::= \textit{and Filter Filter} \mid \textit{or Filter Filter} \\
&\quad \mid = A \; V \mid \textit{!=} \; A \; V \mid > A \; V \mid < A \; V \\
&\quad \mid >= A \; V \mid <= A \; V \mid \textit{like A V} \mid \textit{not\_like A V} \\
&\quad \mid = A \; R \mid \textit{!=} \; A \; R \mid > A \; R \mid < A \; R \\
&\quad \mid >= A \; R \mid <= A \; R \mid \textit{in A VR} \mid \textit{not\_in A R} \\
\textit{Order} &::= \textit{des A} \mid \textit{asc A} \\
\textit{Superlative} &::= \textit{des A V} \mid \textit{asc A V} \\
A &::= \textit{max C T} \mid \textit{min C T} \mid \textit{count C T} \\
&\quad \mid \textit{sum C T} \mid \textit{avg C T} \mid \textit{none C T} \\
&\quad \mid \textit{max MathA} \mid \textit{min MathA} \mid \textit{count MathA} \\
&\quad \mid \textit{sum MathA} \mid \textit{avg MathA} \mid \textit{none MathA} \\
\textit{MathA} &::= + \; A \; A \mid - \; A \; A \mid \times \; A \; A \mid \div \; A \; A \\
C &::= \textit{column} \\
T &::= \textit{table} \\
V &::= \textit{value}
\end{aligned}
$$

Figure 11: The extended grammar for SemQL.

E Dataset Statistics From Spider

Table 5 shows the statistics of our dataset and other cross-domain datasets in the way of Spider. We provide enough examples for both advanced SQL clauses and the calculation type.

F The extended grammar of SemQL

We extend the grammar used in IRNet model to accommodate DuSQL, as shown in Figure 11. The Figure 8 shows the main changes.

Components	Pseudo Descriptions
SQL intersect SQLs	SQL, as set1, SQLs, as set2, belong to set1 and set2
SQL union SQLs	SQL, as set1, SQLs, as set2, belong to set1 or set2
SQL except SQLs	SQL, as set1, SQLs, as set2, belong to set1 but not belong to set2
select A ... A	list A, ... and A
where Conditions	when Conditions
group by C	for each C
group by C having Conditions	the C that Conditions
group by C OrderC	the C with OrderC
order by C Dir	sorted by C Dir
order by C Dir limit value	the top value sorted by C Dir
order by A Dir limit value	the top value sorted by A Dir
A op value	A op value
A op SQL	SQL as v1, A op v1
agg C	agg C
count *	the number of table
T1.C + T2.C	the sum of T1.C and T2.C
T1.C − T2.C	the difference between T1.C and T2.C
T1.C ∗ T2.C	the product of T1.C and T2.C
T1.C / T2.C	T1.C is times of T2.C

Figure 12: The pseudo descriptions for all production rules.