UMEÅ UNIVERSITY

AUTOMATED SQL QUERY GENERATION

RDBMS Testing

Fadi Marouki

Supervisor: Michael Minock
Examiner: Pedher Johansson

Bachelor Thesis, 15 hp
COMPUTING SCIENCE

2019

Abstract

Manually writing SQL queries for Relational Database Management Systems (RDBMS) testing can be very tedious depending on the database complexity. The focus of this thesis is to develop three approaches to automatically generate SQL queries based on a given database instance. These queries can then be used to evaluate the configuration of a RDBMS. The generated queries are only partial components in RDBMS testing. However, they do reduce the amount of work required to perform such configuration assessment. The three presented approaches generate well-formed and semantically meaningful queries (i.e. queries with no logical contradictions). The first approach only consists of a context-free grammar (CFG). The second uses a CFG with an exclusion list. The third uses a CFG with a binary classification machine learning model. The results show that the binary classification algorithm approach outperforms the other two in terms of generating a higher proportion of semantically meaningful queries.

Acknowledgments

I would like to express my great appreciation to Michael Minock at the Department of Computing Science at Umeå University for his valuable and constructive suggestions during the planning and development of this research work. His willingness to give his time has been very much appreciated.

I would also like to thank Pedher Johansson for his insights into the writing of this thesis.

Finally, I want to thank Suna Bensch for her advice which helped at the beginning of this work.

Contents

1 Introduction

Creating software that is maintainable and scalable needs testing. However, the process of manually creating test-cases can be tedious, which is why test automation is of interest. The benefits of test automation are less repetitive work, cost reductions, and maintainability.

Testing is about making sure that every module in all application's tiers is functioning as expected. Applications are usually split into different tiers, **Figure 1** below displays the most common multitiered architecture, a three-tier [8]. There are several ways to test each physical tier, but the scope of this thesis will be restricted to RDBMS which are used in the Data tier.

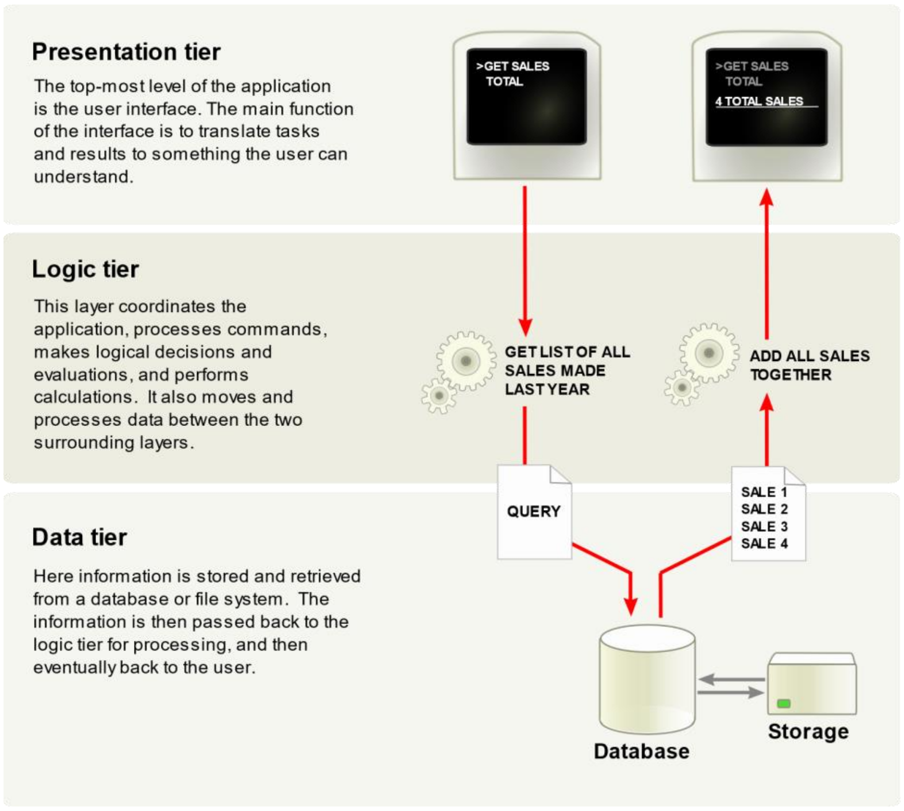
Presentation tier
The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.
>GET SALES TOTAL
>GET SALES TOTAL
4 TOTAL SALES
Logic tier
This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.
GET LIST OF ALL SALES MADE LAST YEAR
ADD ALL SALES TOGETHER
QUERY
SALE 1
SALE 2
SALE 3
SALE 4
Data tier
Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.
Database
Storage

Figure 1. A representation of the most common n-tier application architecture, a three-tier. Image obtained from Wikimedia Commons [2019 March 4].

Ensuring that the data tier is setup properly is extremely important as mistakes may lead to security breaches, unexpected crashes, and data corruption. Although the idea of having a flawless database is straightforward, upholding it in modern databases is a challenging task as they are exceptionally large and complex. Furthermore, Bati, Giakoumakis, Herbert & Surna, 2007 claim that the number of optimizations and ways of execution combined with the expressive power of SQL may result in an unbounded matrix. Moreover, as hardware progresses, DBMS approaches tend to include even more complicated queries along with new functionality to support larger amounts of data. Therefore, evaluating a database engine is a challenging task for now and the future [1].

The configurations involved in RDBMS testing include triggers, functions, updateable views, and transaction isolation levels. Many of these are configured manually with no way of automation. To inspect the values, one may use a Database viewer and manually look through values. However, the database often contains more than a few rows of data. Therefore, SQL queries are required to read, update, insert, or delete data. This is the part that can be automated to ease the amount of work required to validate a database configuration. In this thesis, the generated queries are limited to read queries (SELECT) as described in section 1.1. However, the approaches presented can scale and support different type of queries such as delete, insert and update queries.

Evaluating the generated queries will be done using semantical meaningfulness, i.e. a query with no logical contradictions. It is also possible to expand and create a much more complex criterion. The more complex criteria, the more costly it is to evaluate the queries. Therefore, it is important that the approaches support a training step to minimize the cost of the evaluation.

In this thesis the first approach will represent the foundation to generate SQL queries without any regard in generating semantically meaningful queries. Second approach will aim to improve the amount of semantically meaningful queries by using substrings to create an exclusion list. The third will also aim on improving the number of semantically meaningful queries by using a binary classification algorithm during a training period. Thereafter, evaluate which of the proposed approaches performs better in terms of semantical meaningfulness.

1.1 Delimitations

As SQL queries can be overly complex, only queries with the following structure will be considered here:

```
QUERY ::= SELECT FROM WHERE
SELECT ::= ‘SELECT’*
FROM ::= ’FROM’ (table | table JOIN table)
WHERE ::= ’WHERE’ (term operator value)+
```

Whilst the approaches presented can be applied to any given database instance, **Figure 2** displays the schema used to implement the approaches and produce results. Each entity represents a table and its columns. The arrows between entities denote foreign keys. However, all attributes are ‘text’ data type.

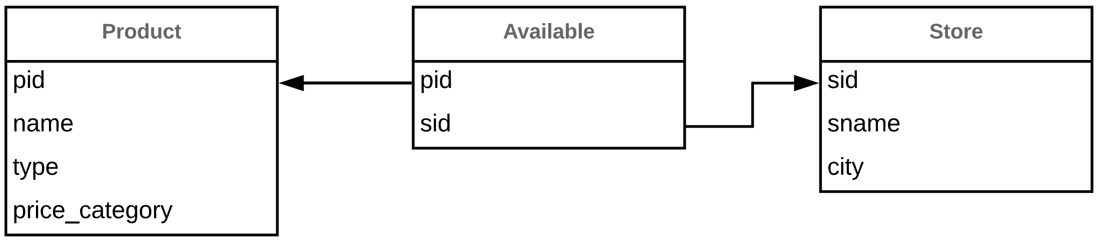

Figure 2. The entity–relationship model used to create queries for. The arrows denote foreign keys. Further on, each entity represents a table and its columns.

Following are an example of SQL expressions which the proposed approaches will evaluate for the database instance model displayed in **Figure 2**:

- Simple query. Product info for product with pid = 1.

  ```
  SELECT X0.* FROM Product AS X0 WHERE X0.pid = '1'
  ```

- A complex query with contradictions. The contradiction is highlighted with an underline and red color.

  SELECT X0.* FROM Available AS X2, Product AS X0 WHERE X2.pid = X0.pid AND <u>X2.pid = '2' AND X2.pid = '3'</u> AND X0.type = 'Soda' AND X0.name = 'Sprite'

- A complex semantically meaningful query.

  ```
  SELECT X0.* FROM Available AS X2, Product AS X0 WHERE X2.pid = X0.pid AND X2.pid = '2' AND X0.name = 'Pepsi'
  ```

2 Background

One of the most straightforward approaches to this problem is to stochastically generate SQL statements. Stochastic parse tree is a tree consisting of SQL syntax tokens built stochastically as the tree is walked. A system called RAGS (Random Generator of SQL) which uses a stochastic parse tree to create complex SQL statements was built for Microsoft SQL Server as an experiment. However, the issues with this system is that the more complex the generated queries were, they either did not return any results or terminate early due to logical contradictions [9].

Another approach is creating SQL mutants that are candidates and then eliminating faulty candidates based on a given SQL statement [10, 11]. This approach requires the user to enter a query and thereafter the system performs small modifications, also called mutations, to generate more queries. The small modifications include adding aggregate functions and other SQL syntax to increase the number of queries. However, this can only produce a limited number of similar queries.

Neither of the approaches address generation of many semantically meaningful queries. Moreover, they are not scalable in the form of handling additional criteria, such as database instance answer returning criterion, i.e. verifying if the generated query has conditions that leads to a result in the given database instance. Therefore, new scalable approaches must be presented that can satisfy multiple criterion.

Brute-force method to construct queries exist as an alternative, however, the amount of redundant and invalid queries makes this approach very unpractical, even though it theoretically results in a total test coverage [7]. Therefore, the main subject of this thesis will be to present three alternative approaches to automatically generate SQL queries.

2.1 Context-Free Grammar

Central to our basic approach here is automata theory. Introduced by the three mathematicians Axel Thue, Alan Turing and Emil Post in the 1930s is what sparked the creation of the first electronic automatic calculator device in 1943. Thus, the foundation of our current electronical devices' dates to the 1930s [4]. Further on, in the 1950s Noam Chomsky introduced Formal Language Theory (FLT) which is the study of syntactical aspects of languages. FLT builds on the earlier work of automata theory and is therefore an especially important field in computing science.

FLT's classical formulation [3], called the *Chomsky hierarchy* has four levels of increasing complexity: regular, context-free, context-sensitive and recursively enumerable languages (**Figure 3**). Each language in FLT is a set of sequences, or strings over some finite vocabulary ∑. Moreover, FLT deals with formal languages that can finitely be defined while the language itself is infinite. Grammars are used to give such finite description of a language. A grammar consists of a finite quadruple (∑, *N, S, R*). ∑ denotes the vocabulary (referred to as *terminals), N* is a second vocabulary of extra symbols (referred to as *non-terminals*), *S* is a special non-terminal that denotes the *start symbol* and R is a set of rules [6].

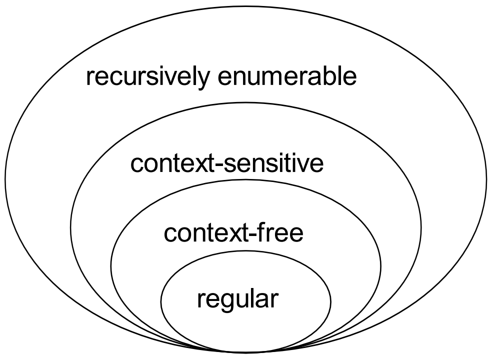

Figure 3. The Chomsky hierarchy, displaying four levels of increasing language complexity. Image obtained from Wikipedia [2019 April 17].

Context-free grammars can be used to construct parse trees using the derivations done from the start symbol. The yield of such parse trees is always a terminal string (**Figure 4**). These trees have proved extremely useful as they clearly show how the symbols of a terminal string are grouped into substrings, each belonging to one of the variables in the language grammar. This has resulted in them being beneficial within compilers. In a compiler the tree structure of the source program facilitates the translation of the source program into executable code by allowing natural recursive functions to perform the translation process. As an example of practical use, context-free grammar describes Document Type Definition (DTD) which sets the rules for allowed syntax in XML (Extensive Markup Language). Therefore, DTD is an essential part of XML [5].

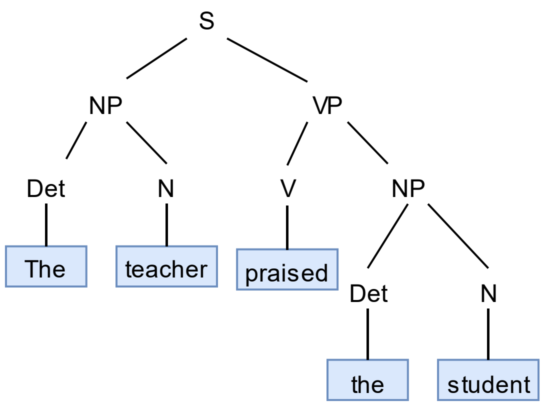

Figure 4. A CFG parse tree that has a terminal string yield.

3 Approaches

This section will describe the base and details of the three proposed approaches for automated query generation. The proposed approaches are based on context-free grammar. A set of rules consisting of terminals and non-terminals describe the SQL syntax described in section 1.1. The SQL syntax is fixed, resulting in the grammar generating well-formed queries, which means that all queries generated can run on a RDBMS without any syntactical errors. The grammar is implemented in Python programming language using the NLTK library. The library provides a way to construct the CFG grammar defined as a string and many other features such as sentence generation, parsing and creating n-grams [2]. To create the grammar, there exist a set of fixed grammar rules that define the overall structure of the generated queries. However, most of the grammar rules are dynamically generated using the database schema defined in XML format. Database values in the XML encoded schema is also supported, and in fact required to generate queries with specific conditions. CFG has the property of generating infinite sentences. This occurs when recursive rules are involved. To eliminate the possibility of not halting when generating queries, a maximum depth of 9 has been set. There is no theoretical idea behind this number, but it is a reasonable depth to generate enough relevant queries.

3.1 Baseline CFG Approach

The first approach is to only use the Baseline context-free grammar (**Figure 5**), including both the fixed and dynamically generated grammar, to generate randomized queries. Thus, meaning that the generated grammar rules are used without any consideration to contradictory queries.

```
S -> SEL XSTAR FR F
SEL -> 'SELECT'
XSTAR -> 'X0.*'
FR -> 'FROM'
F -> ProductRel | StoreRel | AvailableRel

ProductRel -> Product AS ProductVar | Product AS ProductVar WHR ProductWC
Product -> 'Product'
ProductWC -> ProductWCVal | ProductWCVal GATE ProductWC
ProductWCVal -> ProductVar Product_pid OPR ProductpidAttrVal | ProductVar Product_name OPR
ProductnameAttrVal | ProductVar Product_type OPR ProducttypeAttrVal | ProductVar
Product_price_category OPR Productprice_categoryAttrVal

Product_pid -> '.pid'
Product_name -> '.name'
Product_type -> '.type'
Product_price_category -> '.price_category'
ProductpidAttrVal -> "'1'" | "'2'" | "'3'"
ProductnameAttrVal -> "'Coke'" | "'Pepsi'" | "'Sprite'"
ProducttypeAttrVal -> "'Soda'"
Productprice_categoryAttrVal -> "'Drinks'"
```

Figure 5. A part of the CFG rules dynamically generated from a database xml schema.

3.2 CFG with Exclusion List

This approach utilizes the baseline CFG above, but it aims on improving the rate of semantically meaningful queries by creating an exclusion list which is meant to work as a filterer. The exclusion list is created during a training period where a set of queries are ran by the satisfiability checker and split into substrings of fixed length N (for some N), also called *n-grams*. If contradictions are present in a generated query, its n-grams are added to the exclusion list and if the generated query is satisfiable, its n-grams are removed from the exclusion list should they be present. However, this introduces false positives, like the example shown in **Figure 6**.

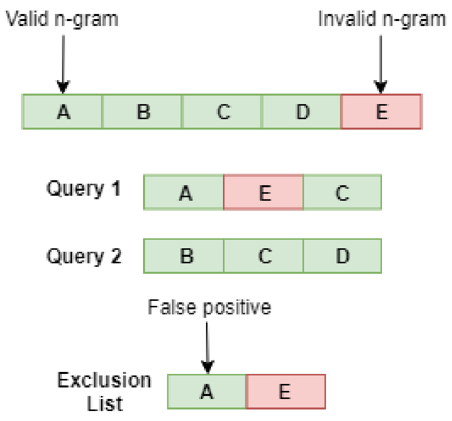

Figure 6. Two queries consisting of valid and invalid n-grams. The exclusion list is trained on these queries, but a false positive is created.

Moreover, this fragment exclusion list-based approach also requires a value for N to determine the size of the substrings. Therefore, this will be carefully analyzed and picked to give the most accurate results with highest percentage of semantically meaningful queries.

3.3 CFG with Binary Classification

The third presented approach is to use the CFG grammar combined with a binary classification. Binary classification is a machine learning algorithm that uses data to determine the category, type, or class of an item or row of data.

This model is implemented in C# programming language using Microsoft's .NET Core framework. Moreover, binary logistic regression classification is the model used [12]. To train this model, a stochastic dual coordinate accent method is used. Furthermore, the data used to train the model is a list of queries combined with a 0/1 depending if they are semantically meaningful. The model makes thereafter prediction based on test data, which is why the more it is trained, the more accurate it becomes (**Figure 7**).

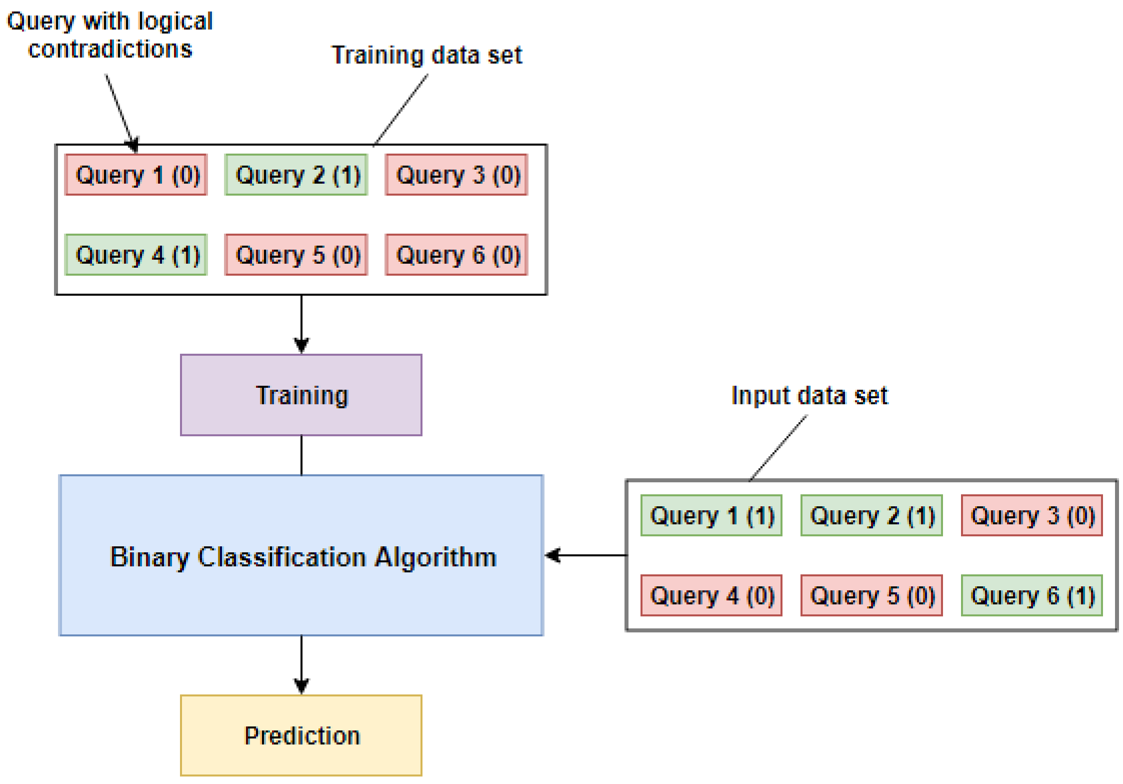

Figure 7. Illustration over the main properties of the binary classification algorithm.

4 Approach Evaluation

Each of the three proposed approaches are analyzed in terms of their own properties and the general defined criterion, semantic meaningfulness. Different number of queries will be generated multiple times for each approach to obtain the average accuracy percentage. This gives an approximate picture of how the suggested approaches scale with number of queries. Further on, to analyze each generated query, a built-in satisfiability checker in an external tool called C-Phrase (https://c-phrase.com) will be used. The checker is built using the natural language toolkit (NLTK) library's theorem prover, Prover9, with some string manipulation helper functions to provide the required arguments.

The number of queries generated start with 10 and then step by 10 up to 100 queries. Thereafter, the number of queries is stepped by 100 up to 1000 queries. The 10 stepping at the beginning is set to give an accuracy overview when generating a low number of queries. The maximum of 1000 is set to give a general idea of how the approach performs when generating many queries. Moreover, the total set of queries is not large enough to support query generation over 1000 due to set limitations in section 1.1. In addition, each approach does four generations for every query count to give an average percentage.

4.1 Baseline CFG

The Baseline CFG method does not have any training or additional arguments. Due to this fact, it is expected that the semantic meaningfulness percentage may be low. As expected, **Figure 8** displays a low semantical meaningful percentage for all query counts. Considering only a single criterion is taken into account, this percentage is too low to be considered useful.

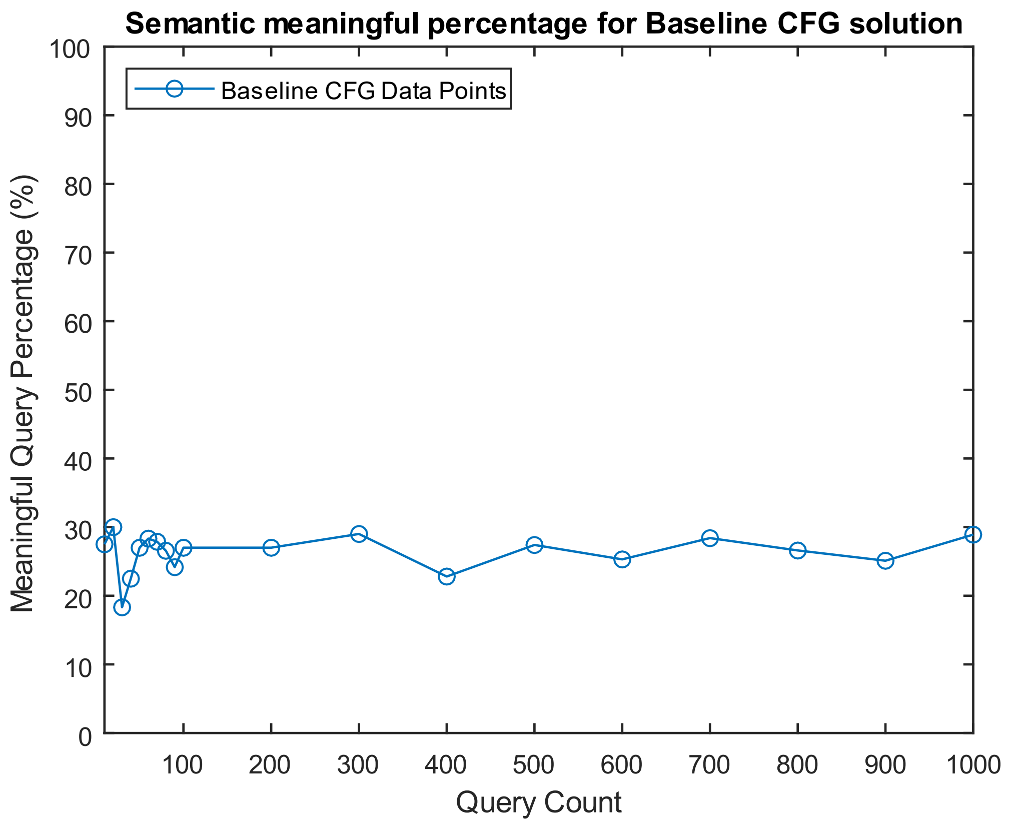

Figure 8. Semantic meaningful percentage for Baseline CFG approach. Each data point represents a test conducted with a set number of queries to generate.

4.2 CFG with Exclusion

CFG with Exclusion List is based on n-grams, or more specifically substrings of a query. Its training period consist of analyzing equal number of queries it must generate. However, to determine a reasonable substring size to train on, a test was performed in an equivalent way to query generation, but semantic meaningfulness for a substring size was the parameter being measured. It is expected that the semantic meaningfulness will be low for small and big n-gram sizes, that assumption is based on the simple fact that short and long substrings are not good enough unique identifiers. **Figure 9** shows that the semantic meaningfulness is best around substring size of 7-9, but the highest percentage is observed at the size 9. Therefore, this n-gram size is used in the semantic meaningfulness test with different number of queries to generate.

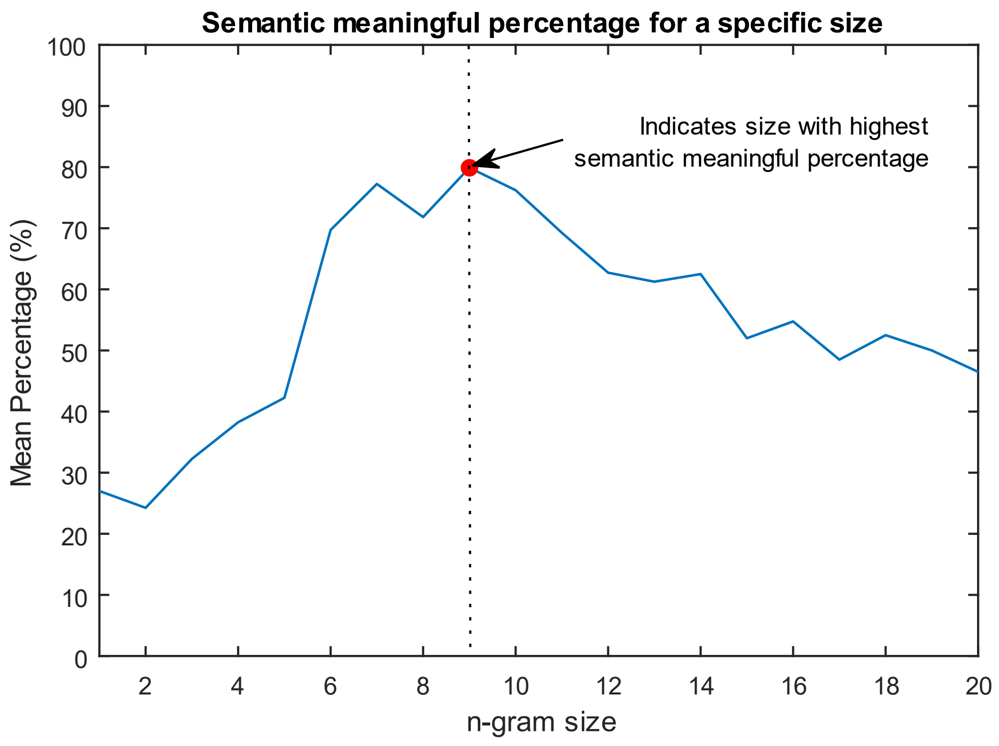

Figure 9. Semantic meaningful percentage for different n-gram sizes for the CFG with Exclusion List approach. The red dot indicates the highest semantic meaningful percentage of 79.9% with the n-gram size 9.

Figure 10 displays the semantic meaningful percentage for n-gram size 9. The results show that the approach is inaccurate at small number of queries which is expected because the methods train on equal number of queries they must generate. However, even though the semantic meaningful percentage is much higher than what is observed in Baseline CFG approach, the approach shows a decline in meaningful percentage for higher number of queries. The decline could be due to false positives which is described in section 3.2, or due to the simple fact that substrings are not good unique identifiers.

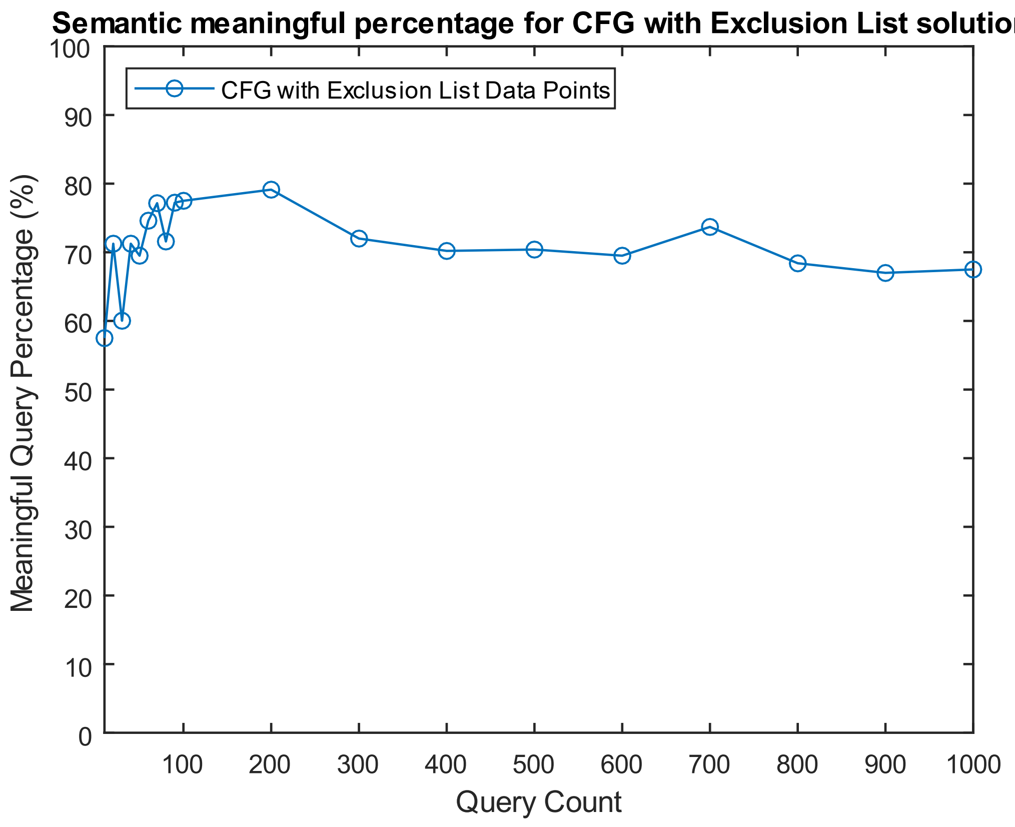

Figure 10. Semantic meaningful percentage for CFG with Exclusion list approach. Each data point represents a test conducted with a set number of queries to generate.

4.3 CFG with Binary Classification

CFG with Binary Classification has a training period where it trains with an equal number of queries it must generate. **Figure 11** shows that the approach is inaccurate for small number of queries. However, it displays an upgoing trend of semantic meaningfulness for higher query counts.

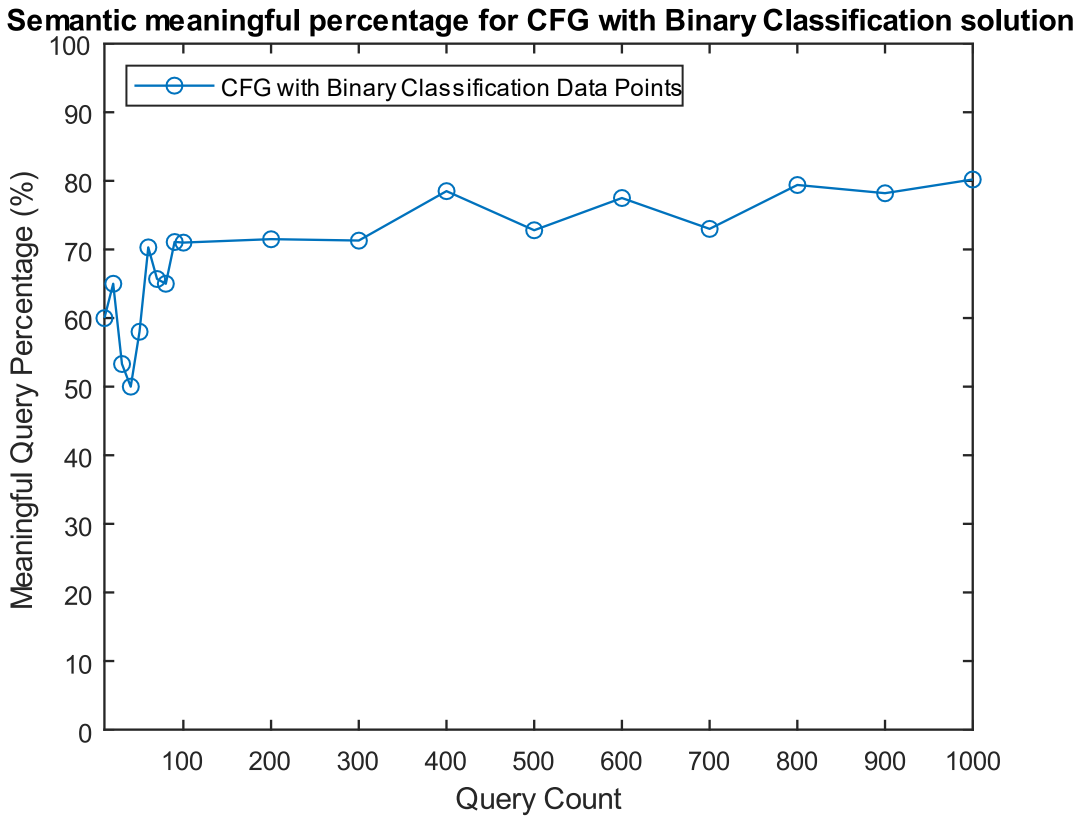

Figure 11. Semantic meaningful percentage for CFG with Binary Classification approach. Each data point represents a test conducted with a set number of queries to generate.

4.4 All Approaches

Both Exclusion List and Binary Classification based approaches show a definite increase in semantic meaningful percentage in comparison to the Baseline CFG. Moreover, the binary classification approach provides a higher semantic meaningful percentage for larger query counts than the exclusion list approach (**Figure 12**).

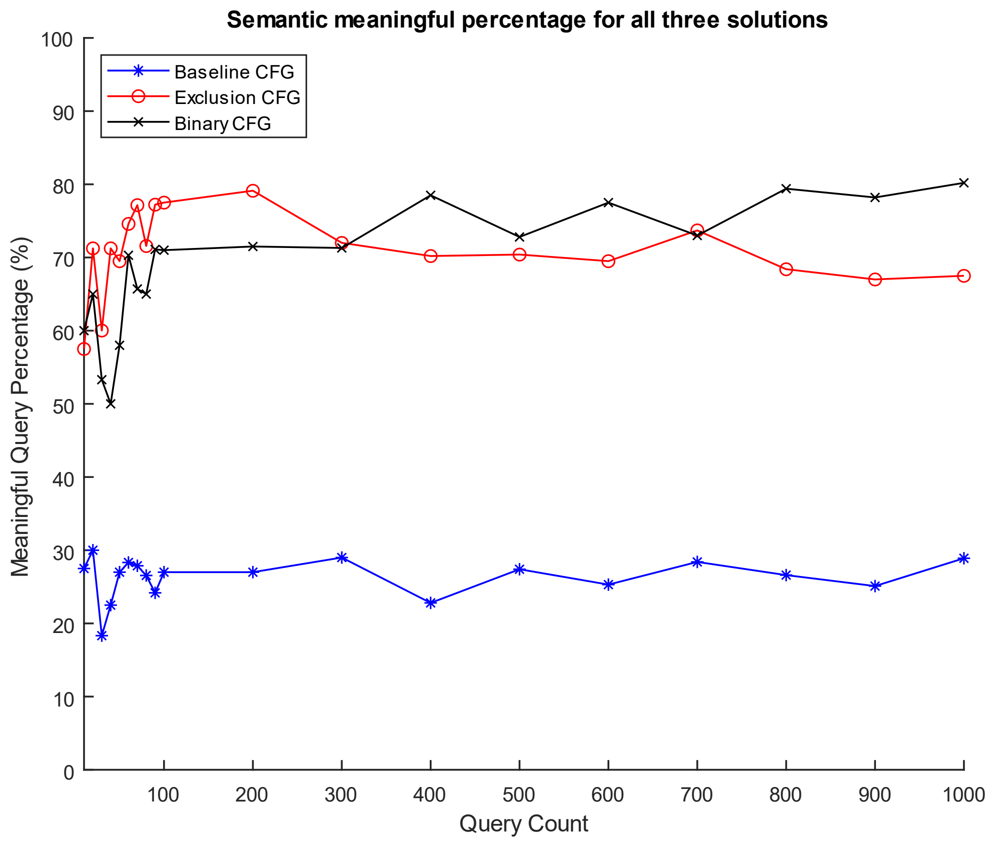

Figure 12. Displays all three approaches semantic meaningful percentage.

5 Discussion

All three approaches show a wide variance in results when it comes to generating queries with a number between 10-100. For the Baseline CFG approach, it is random as it does not have a training period. However, for the other two approaches, utilizing a training period shows a significant improvement in comparison to the Baseline CFG even with little training. The approaches train with equal number of queries they must generate. Therefore, the training these methods perform is not stable until around 100 queries, where the high volatility in semantic meaningful percentage cease to exist.

Baseline CFG generated queries with a low semantic meaningful percentage. This result is what was expected because it has no training and therefore no way of improving its results. However, it provides a solid foundation to build upon it and that is exactly what the other two approaches do.

The Exclusion List based approach utilize substrings and attempt to create a database of bad substrings to distinguish between good and bad queries. This approach shows quite remarkable improvements to the results of the Baseline CFG. However, the data shows that the semantic meaningful percentage goes down with higher number of queries to generate. This indicates that the method is unstable and has no room for improvement with more training. In addition, this approach uses a lot of memory to construct its exclusion list.

The Binary Classification algorithm shows a definite increase in semantic meaningful queries with more training. Moreover, it has a max of 80.2% semantic meaningfulness compared to the exclusion list-based approach with a max of 79.13%. Moreover, the binary classification is expected to be more practical. The exclusion list approach might become unworkable should an exponential number of items need to be stored in memory. Therefore, the conclusion drawn in this thesis is that the binary classification method outperforms the other two in terms of being solid and semantic meaningful scalable.

5.1 Comparison to Earlier Methods

The two issues, scalability and query quantity, mentioned in the earlier methods in Background are the biggest factors to why earlier methods fail on solving the task of generating a vast amount of SQL queries. With scalability, they fail to address the possibility in generating queries that fulfill semantic meaningfulness or any other criterion. Neither are they able to support the option to add additional criteria due to the way they are constructed. That alone is a reason RAGS system failed [9]. Moreover, whilst mutation SQL query generation method has not officially been discontinued; it is not fit to generate a vast amount of SQL queries with semantic meaningfulness in mind [10, 11].

Binary classification however is scalable in form being able to train on multiple criteria such as semantic meaningfulness and database instance answer returning. Moreover, it can generate a large number of SQL queries should the grammar be large enough. Therefore, the presented approach is far better than earlier methods.

6 Conclusions

In this thesis, three different approaches were proposed to generate large number of SQL queries with semantic meaningfulness as a focus. Syntactic meaningfulness was ignored as the foundation of the approaches generates well-formed SQL. The first approach, Baseline CFG, provides the foundation of query generation with no training to improve the results. However, the second and third approach build on top of the first one with a training step to improve the results in terms of the proportion of semantically meaningful queries. The second one utilizes substrings and builds and exclusion list whilst the third one utilizes a binary classification algorithm to train.

Results show that the Baseline CFG approach has a low semantic meaningfulness and that itself is self-explanatory on why a training step is required. The exclusion list-based approach performed almost same as the binary classification algorithm; however, it fell short on the scalability part. In terms of generating a vast number of SQL queries, the binary classification algorithm is what we would recommend.

The specific use of this work is to automate a certain part of RDBMS testing and performance profiling by eliminating the need to manually write queries over a given database schema. Such automation is especially useful when the given schema is complex with many constraints.

6.1 Future Work

The findings in this thesis opens doors for future improvements to get a functional product into the industry. Demonstrated generation of SQL queries was over a small subset of the SQL grammar. However, this is of no issue and can easily be expanded in the proposed approaches. Moreover, models that support a training period to allow generation of queries that follow specific criteria. The criterion introduced here is semantic meaningfulness, which was defined as a query with no logical contradictions. However, it might be more applicable to add database answer generation baseline criterion as well, i.e. a query returns results for a specific database instance. The used criterion, e.g. semantically meaningfulness, is just one of many criterions that can be used to filter and shape the generated queries. The criteria for the generated queries may be much more complex than the one defined in this thesis.

Moreover, even though the binary classification algorithm proved that it can generate quite accurate results with a little training. There is room to advance the training step to improve its accuracy even further. This study only introduced the read type of queries. However, as previously stated, it may be expanded to support other types e.g. insertion, deletion, and update, but under more suitable criteria. This can also be expanded to check the approaches for resource usage.

References

[1] Hardik Bati, Leo Giakoumakis, Steve Herbert, and Aleksandras Surna. A genetic approach for random testing of database systems. In Proceedings of the 33rd international conference on Very large data bases, pages 1243–1251. VLDB Endowment, 2007.

[2] Steven Bird, Ewan Klein, and Edward Loper. Natural language processing with Python: analyzing text with the natural language toolkit. " O'Reilly Media, Inc.", 2009.

[3] Noam Chomsky. Three models for the description of language. IRE Transactions on information theory, 2(3):113–124, 1956.

[4] G. T. Q Hoare. 1936: Post, turing and 'a kind of miracle' in mathematical logic. The Mathematical Gazette, 88(511):2–15, 2004.

[5] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. Context-Free Grammars and Languages, pages 171–194. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[6] Gerhard Jäger and James Rogers. Formal language theory: refining the chomsky hierarchy. Philosophical Transactions of the Royal Society B: Biological Sciences, 367(1598):1956–1970, 2012.

[7] Shadi Abdul Khalek and Sarfraz Khurshid. Automated SQL query generation for systematic testing of database engines. In ASE, pages 329–332. Citeseer, 2010.

[8] Heiko Schuldt. Multi-Tier Architecture, pages 1862–1865. Springer US, Boston, MA, 2009.

[9] Donald R Slutz. Massive stochastic testing of SQL. In VLDB, volume 98, pages 618–622. Citeseer, 1998.

[10] Javier Tuya, Ma Jose Suarez-Cabal, and Claudio De La Riva. Sqlmutation: A tool to generate mutants of SQL database queries. In Second Workshop on Mutation Analysis (Mutation 2006-ISSRE Workshops 2006), pages 1–1. IEEE, 2006.

[11] Javier Tuya, Ma José Suárez-Cabal, and Claudio De La Riva. Mutating database queries. Information and Software Technology, 49(4):398–417, 2007.

[12] Tutorial: Analyze sentiment of website comments with binary classification in ML.NET, URL https://docs.microsoft.com/en-us/dotnet/machine-learning/tutorials/sentiment-analysis. Accessed [2019 05 15].