Automatically Synthesizing SQL Queries from Input-Output Examples

MohammadAmin Raeisi

Summary

This paper proposes a SQL synthesizer for the first time that produces SQL queries from I/O examples given by end-users and is implemented as SQLSynthesizer tool. The previous methods of generating SQL queries required sufficient knowledge of SQL DSL which is not suitable for non-expert end-users. But SQLSynthesizer fully automatically infers a SQL query that satisfies the given I/O examples.

The main idea of SQLSynthesizer consists of three parts: First it creates an incomplete SQL query (called a query skeleton) from Input examples to capture the basic structure of the result query. Second it completes these query skeletons by inferring query conditions as learning appropriate rules that can perfectly divide a search space into a positive and a negative part. Also by searching for aggregates for every column in the output table that has no matched column in the input table, and outputs a list of queries that satisfy the provided example input and output. Finally uses the Occam's razor principle to rank simpler queries higher in the output.

Creation skeletons consists of three parts itself: First one is Query tables, which SQLSynthesizer uses a heuristic to estimate the query tables. If the same column from an input table appears N times in the output table, SQLSynthesizer replicates the input table N times in the query table set. Second one is join conditions, which SQLSynthesizer First seeks to join tables on columns with the same name and data type. If such columns do not exist, SQLSynthesizer seeks to join tables on columns with the same data type (but with different names). And the last one is projection columns, which SQLSynthesizer scans each column in the output table, and checks whether a column with the same name exists in an input table. If so, SQLSynthesizer uses the first matched column from the input table as the projection column. Otherwise, the column in the output table must be created by using an aggregate.

Strengths

- This is the first attempt to write a synthesizer in SQL DSL.
- It expresses the motivation well without adding any extra words in the introduction part.
- To simplify the problem and prune the search tree, keywords that can be inferred from other main keywords have been removed.
- It shows the SQLSynthesizer's workflow well with a diagram.

Weaknesses

- In the example presented at the beginning, this article should explain its algorithm to reach the output from the table instead of just saying what the output should be.
- The explanations related to part Online survey, which is given at page 3, should come under the title experiments at the end.
- The font of the titles should change to make it easier to find different parts of the article.
- Before introducing the DSL, it's better to explain the algorithm with an example to illustrate the solution.
- It supports few features of the SQL language.
- It's better to bring the pseudo-code of all methods used in the algorithm.
- It doesn't explain the algorithm in detail and only mentions the generals.
- It is better to test more than 28 samples for benchmarking.
- It doesn't specify the cost of the operators to give queries a ranking.