

**A Model for Automatic Partial Evaluation of SQL Queries**

by

Fauhat Ali Khan Panni

MASTER OF ENGINEERING





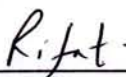
Department of Computer Science and Engineering

BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY

January 2018

The project report titled “A Model for Automatic Partial Evaluation of SQL Queries”, submitted by Fauhat Ali Khan Panni, Roll No. **0412052005 F**, Session April 2012, to the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, has been accepted as satisfactory in partial fulfillment of the requirements for the degree of Master of Engineering in Computer Science and Engineering and approved as to its style and contents. Examination held on January 28, 2018.

### **Board of Examiners**

1.   
\_\_\_\_\_
- Dr. Abu Sayed Md. Latiful Hoque  
Professor  
Department of Computer Science and Engineering  
Bangladesh University of Engineering and Technology  
Chairman  
(Supervisor)
  
2.   
\_\_\_\_\_
- Dr. M. Sohel Rahman  
Head and Professor  
Department of Computer Science and Engineering  
Bangladesh University of Engineering and Technology  
Member
  
3.   
\_\_\_\_\_
- Dr. Rifat Shahriyar  
Assistant Professor  
Department of Computer Science and Engineering  
Bangladesh University of Engineering and Technology  
Member

# Candidate's Declaration

This is hereby declared that the report titled "A Model for Automatic Partial Evaluation of SQL Queries" is the outcome of the project work carried out by me under the supervision of Prof. Dr. Abu Sayed Md. Latiful Hoque, in the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka. It is also declared that this report or any part of it has not been submitted elsewhere for the award of any degree or diploma.



---

Fauhat Ali Khan Panni

Candidate

# Acknowledgment

First of all, I would like to express my sincere gratitude to my supervisor Prof. Dr. Abu Sayed Md. Latiful Hoque for his continuous guidance, direction and support without which this project could not have been completed. I would like to thank everyone who scored SQL answers in the data sets. Without their kind help, experimentation for this project would not have been possible. I would like to thank the examiners of my project report, Prof. Dr. M. Sohel Rahman and Dr. Rifat Shahriyar for their valuable comments and suggestions. I would also like to thank my family for helping and supporting me throughout my postgraduate study.

# Abstract

Database is the core course in the study of Computer Science and Engineering. In the undergraduate level, one of the major topics for database is SQL. SQL-LES is a Problem-Based e-Learning (PBeL) system that has been being used in learning and teaching of SQL in undergraduate level. In SQL-LES, students submit SQL answers in online examinations and assignments. This type of systems with auto-evaluation feature typically perform evaluation by comparing the result-sets returned by the answered SQL expression and the correct expression stored in the system for the respective problem. This approach of evaluation based on result-set comparison gives full marks when the results match and zero otherwise. This creates a frustration to the students whose answers are almost correct and is evaluated to zero grade.

In this report, we introduce a model for evaluating the partially-correct SQL answers. The key idea is to calculate a score based on the syntactic similarity between the answered SQL expression and the correct SQL expression for the respective problem. In many cases there can be more than one correct SQL expression. This issue is addressed by calculating a score with respect to every correct expression and assigning the maximum score to the answer. As evaluation based on comparison of result-set ensures full scores to SQL answers that are completely correct, this approach can be utilized to filter out the correct SQL answers and after filtering them out, the partial-evaluation model can be applied to score only the answers which are not completely correct.

We experimented our model for different real life data sets obtained from database practical course. In these data-sets, we obtained the human score by database teachers. We have compared the score generated by the model and human score. The comparison result is found to be quite satisfactory.

# Contents

|   |            |
|---|------------|
| <b>Board of Examiners</b>                           | <b>i</b>   |
| <b>Candidate's Declaration</b>                      | <b>ii</b>  |
| <b>Acknowledgment</b>                               | <b>iii</b> |
| <b>Abstract</b>                                     | <b>iv</b>  |
| <b>1 INTRODUCTION</b>                               | <b>1</b>   |
| 1.1 Background . . . . .                            | 1          |
| 1.2 Problem Definition . . . . .                    | 2          |
| 1.3 Objectives and Outcome . . . . .                | 2          |
| 1.4 Overview of the Project . . . . .               | 3          |
| 1.5 Organization of the Project Report . . . . .    | 4          |
| <b>2 Related Works</b>                              | <b>5</b>   |
| 2.1 Automatic Evaluation in Programming . . . . .   | 5          |
| 2.2 Automatic Evaluation in Database . . . . .      | 7          |
| 2.3 Similarity between Nominal Attributes . . . . . | 8          |
| <b>3 Partial Evaluation Model</b>                   | <b>9</b>   |
| 3.1 Analysis of SQL Statement . . . . .             | 9          |
| 3.2 Scoring SELECT Clause . . . . .                 | 14         |
| 3.3 Scoring FROM Clause . . . . .                   | 15         |
| 3.4 Scoring WHERE Clause . . . . .                  | 17         |
| 3.5 Scoring GROUP BY Clause . . . . .               | 22         |
| 3.6 Scoring HAVING Clause . . . . .                 | 23         |
| 3.7 Scoring ORDER BY Clause . . . . .               | 26         |
| 3.8 Overall Score . . . . .                         | 27         |
| 3.9 Scoring Multiple Queries . . . . .              | 27         |

|          |                                  |           |
|----------|----------------------------------|-----------|
| <b>4</b> | <b>Experimental Results</b>      | <b>29</b> |
| 4.1      | Evaluation Methodology . . . . . | 29        |
| 4.2      | Model Evaluation . . . . .       | 31        |
| <b>5</b> | <b>Conclusion</b>                | <b>40</b> |
|          | <b>Appendices</b>                | <b>44</b> |

# List of Figures

|     |   |    |
|-----|---|----|
| 3.1 | Structure of an SQL DML Expression . . . . .  | 10 |
| 3.2 | Structure of SQL SELECT Clause . . . . .  | 10 |
| 3.3 | Structure of SQL FROM Clause . . . . .  | 11 |
| 3.4 | Structure of SQL WHERE Clause . . . . .   | 11 |
| 3.5 | Structure of SQL GROUP BY Clause . . . . .  | 12 |
| 3.6 | Structure of SQL HAVING Clause . . . . .  | 12 |
| 3.7 | Structure of SQL ORDER BY Clause . . . . .  | 13 |
| 4.1 | Comparison between scores by our model and 1st expert for data-set 1 . . . . .  | 32 |
| 4.2 | Chart for similarity and dissimilarity counts of score (in percentage) by 1st expert<br>and our model for data-set 1 . . . . .  | 33 |
| 4.3 | Comparison between scores by our model and experts for data-set 2 . . . . .   | 34 |
| 4.4 | Chart for similarity and dissimilarity counts of scores (in percentage) by 2nd expert<br>and our model for data-set 2 . . . . . | 35 |
| 4.5 | Chart for similarity and dissimilarity counts of scores (in percentage) by 3rd expert<br>and our model for data-set 2 . . . . . | 36 |
| 4.6 | Comparison between scores by our model and 3rd expert for data-set 3 . . . . .  | 37 |
| 4.7 | Chart for similarity and dissimilarity counts of scores (in percentage) by 3rd expert<br>and our model for data-set 3 . . . . . | 37 |
| 4.8 | Comparison between scores by our model and 3rd expert for data-set 4 . . . . .  | 38 |
| 4.9 | Chart for similarity and dissimilarity counts of scores (in percentage) by 3rd expert<br>and our model for data-set 4 . . . . . | 39 |
| A.1 | Database Schema for Data Set 1 . . . . .  | 48 |
| A.2 | Database Schema for Data Set 2 and Data Set 4 . . . . .   | 49 |
| A.3 | Database Schema for Data Set 3 . . . . .  | 49 |



# List of Tables

|     |   |    |
|-----|---|----|
| 4.1 | Types of SQL Query Expressions . . . . .  | 30 |
| 4.2 | Summery of Data . . . . .   | 30 |
| 4.3 | Similarity Categories . . . . .   | 31 |
| 4.4 | Values used for different variables in partial evaluation model . . . . .   | 32 |
| 4.5 | Table for similarity and dissimilarity counts of scores (in percentage) by 1st expert<br>and our model for data-set 1 . . . . . | 33 |
| 4.6 | Table for similarity and dissimilarity counts of scores (in percentage) by 2nd expert<br>and our model for data-set 2 . . . . . | 34 |
| 4.7 | Table for similarity and dissimilarity counts of scores (in percentage) by 3rd expert<br>and our model for data-set 2 . . . . . | 35 |
| 4.8 | Table for similarity and dissimilarity counts of scores (in percentage) by 3rd expert<br>and our model for data-set 3 . . . . . | 36 |
| 4.9 | Table for similarity and dissimilarity counts of scores (in percentage) by 3rd expert<br>and our model for data-set 4 . . . . . | 38 |
| A.1 | Data Set 1 . . . . .  | 44 |
| A.2 | Data Set 2 . . . . .  | 45 |
| A.3 | Data Set 3 . . . . .  | 46 |
| A.4 | Data Set 4 . . . . .  | 46 |
| A.4 | Data Set 4 . . . . .  | 47 |
| A.4 | Data Set 4 . . . . .  | 48 |

# Chapter 1

## INTRODUCTION

### 1.1 Background

Problem-based learning (PBL), an interactive learning method originating in medical education is one of the most futuristic and outcome-oriented teaching and learning strategy. Problem-based learning is “an instructional method that initiates students’ learning by creating a need to solve an authentic problem” [1]. Although the method emerged from medical sciences today it is no more a monopoly of any discipline. Rather it is a general approach that is being adopted by various disciplines especially those that are driven by practical and hands-on skills. Computer science is one of the disciplines where problem-based learning is getting increasing attention. Problem-based learning has been employed to teach foundational computer science courses [2]. PBL has also been experimented with for software engineering classrooms [3].

One very useful feature of any type of e-learning system is e-evaluation i.e. automated evaluation of student’s answers. In a typical database problem-based e-learning system, problems that students solve either for assignments or exams are generally SQL queries. A problem-based e-learning system named SQL-LES for conducting practical database course has been described in [4]. The system contains a question bank containing SQL questions and answers of various complexities. Instructors can use the question bank to select questions of different complexity levels and group them together to create test sets for individual students. A challenging problem that today’s ever-evolving PBL systems need to address is automatic evaluation of the solutions submitted to the system by the students. Most of the existing systems provide teachers or evaluators easy to use and flexible user interface to assign evaluated score to individual answers. Most of the systems that support automatic evaluation evaluate answers as either correct or incorrect. But in

most of the cases, the evaluation is still done by the instructor/evaluator and the system merely makes evaluation easy for the evaluator. The problem of automatic evaluation is not an easy one to solve. However, solutions can still be formulated so that the system performs better than before. This project report focuses on automatic evaluation in a PBL (Problem-based e-Learning) system for database courses. The problem this project tries to solve is evaluating and scoring SQL queries answered by students in a database PBL system.

## **1.2 Problem Definition**

In a typical database e-learning system generally few common features are present. The system contains a question bank from which the instructors set questions for assignments and online exams. In most of the cases, the questions are querying a database and the answers are SQL query expressions. E-learning systems that have auto-evaluation features generally evaluate the answered SQL expression as correct or incorrect (or a full mark or zero). Most of the time, the system achieves this by comparing the result set returned by the answered queries and the correct queries stored for the respective questions. In this report we will refer the correct SQL expression stored in the database of the learning system for evaluation as the reference query expression and we will refer the query expression submitted by a student as an answer as the answered query expression. This approach of evaluation based on result set returned by the answered and reference SQLs results in assigning a full score to the answers that are fully correct and assigning a zero to all other answers even if some of them are almost correct. In this project, we address partial evaluation of SQL expressions. We describe partial evaluation as assigning a score to an SQL expression that is not necessarily fully correct. In other words, we address the problem of determining partial scores for those SQL expressions that are partially correct.

## **1.3 Objectives and Outcome**

The objectives of this project report are to:

1. find the score for the students' SQL submission that are partially correct,

2. evaluate the correctness of the model by comparing the scores given by the instructors and the scores obtained by using the model.

The outcome of this project is a model to generate scores for answered SQL expressions whether the answered expression is fully or partially correct.

## 1.4 Overview of the Project

The main objective of this project is to develop model for scoring SQL expressions. This is achieved by making comparison between the answered expression and the reference expression for the respective SQL question. So the key idea behind the scoring is measuring syntactic similarity between the answered expression (the expression that is being scored) and the reference expression for the respective question. The greater the similarity is, the better is the score and it is vice versa. In some cases, the result sets for the answered expression and the reference expression will be an exact match. In that case, even though the score for the answered expression can still be calculated using the model it is more straight forward to assign the full score without calculating it. But in many cases, a near perfect answer may not produce the same result set as the one produced by the reference expression. There are SQL problems that can be solved in more than one way by more than one SQL expressions. In other words, multiple correct answers may exist for one question. In that case, the best solution is to store multiple reference expressions (if they exist) for one SQL problem. The similarity should be measured between the answered expression and every reference expression that exist for that question and assigning the best score among all the scores calculated for different reference expressions to the answer.

To measure the syntactic similarity we analyze all the different clauses an SQL expression can have. We analyze the syntactic structure of each clause and develop scoring patterns for the clauses. There are clauses that are similar to each other and that can be scored using a similar scoring pattern. The scoring of similar clauses follow similar scoring pattern.

So the overall scoring of an SQL expression consists of two basic steps - in the first step all the clauses the answered and/or reference query contains are scored individually and in the second

step all the individual scores are combined to find the overall score of the answered expression. While combining the individual scores of the clauses the complexity of each of the clauses are taken into consideration as a lesser complex clause should have lesser impact to the overall score while a more complex clause should have more impact.

We experiment with our model by applying it to different SQL expressions of data sets and compare the scores with the scores assigned by human experts.

## **1.5 Organization of the Project Report**

The rest of the report is organized as follows:

In chapter 2, we discuss about the learning systems that have automatic evaluation to some extent. Also we discuss a related technique to measure similarity between two objects.

In chapter 3, we present our partial evaluation model in details along with the analysis of the different SQL clauses.

In chapter 4, we present the experimental results for our partial evaluation model.

In chapter 5, we conclude our report with further discussion.

# Chapter 2

## Related Works

In this chapter we discuss a few problem-based e-learning (PBeL) systems for computer programming with auto-evaluation feature. We then discuss different database PBeL systems that accommodate some form of auto-evaluation. We also discuss an existing technique in the literature for measuring similarity or dissimilarity between objects.

### 2.1 Automatic Evaluation in Programming

Kitaya and Inoue [5] presented a web-based system for automatic evaluation of java programming assignments. The system evaluates syntactically-correct student programs that are either console programs or programs that contain multiple classes and/or methods. Syntactical correctness is a requirement for a program to be evaluated. So the programs are first checked for syntactical correctness before evaluating them. If the program being evaluated contains only a main method (i.e. a console program) then the results of the program for at least more than one input are compared with the results of a reference program which is the correct program for the respective assignment. If for all the inputs the results match then the program is considered as correct; otherwise incorrect. If a program contains classes or methods other than main as per the requirement of the respective assignment then first a JUnit test is conducted to verify if the classes and/or methods are written according to the instructions. Then the step which compares the results of the program being evaluated and the reference program is followed to find if the program is correct or not.

Karavirta and Ihantola [6] presented an auto assessment tool for Javascript programming exercises. The open-source tool named js-assess is written in Javascript and HTML and runs on students' browsers. As a result the tool is suitable for self-studying as there is no way to store

students' progresses in a server. The different features of a student's code that can be assessed using the tool are functionality, style, programming errors and different software metrics (such as lines of code, lines of comments, statement count, branch count etc.). There are several open-source Javascript tools that support many of the features listed above. The tool js-assess sits on top of these various tools and combine the functionality of these tools for auto-assessment and provide feedback to the students.

Ala-Mutka et al [7] presented Style++, a tool that automatically assesses styles of students' C++ programs. The tool has been developed and used in the Tampere University of Technology in Finland for evaluating students' programming styles. The goal of the tool is to guide students to write reliable, maintainable and clear object-oriented C++ code while having a clear understanding of correct usage of the risky features of C++. Also for the courses where certain features of the tools are not required, the tool can be configured with a configuration file to turn off certain features. When no configuration is given the tool works with a default configuration.

Buyrukoglu et al [8] proposed a semi-automated evaluation approach for providing feedback to novice programming students. In a semi-automated mode, both the system and the Instructor(s) contribute in evaluation. The study makes observation that even though two programs written by two different programmers solving the same problem are not likely to be identical, but both the programs may contain some common code segments. This can be true for a number of programs solving the same problem. The authors of the study make this observation based on students' codes. In the proposed approach, logically similar code segments are considered components. Two code segments are considered similar if they have similar control structure, similar conditions and similar blocks. In this way, two or more students codes may have many similar components. The examiner can comment all the components just once and all the students will get comments for the respective components that their codes contain. This saves a great amount of time for the instructors while students get prompt comments on their programs.

Singh et al [9] presents a method for providing automated feedback for programming assignments of beginner level programming students. The key components for feedback generation include a reference implementation of the program and an error model that describes the possible

errors that students can make. The automated feedback generation tool provides an error model language that can be used to write possible corrections to errors that students might make while writing a program. The tool explores a set of candidate programs based on the correction rules to the student program and finds a program that requires minimal corrections.

## **2.2 Automatic Evaluation in Database**

Chandra et al [10] presented XData system with partial marking as a key feature. The concept behind their partial marking scheme is computing a score for a student query based on how close it is to the instructor query. This is very similar to our approach to calculate partial scores. However the detailed model containing the functions that calculate the score was not presented in the paper. As our central focus is the model containing the mathematical functions that calculate the partial scores for answered queries the contribution of this report is the mathematics behind the scoring.

SQL-LES [4] is an interactive system for teaching and learning of different components of database. One key feature of the system is an extensive SQL question bank containing SQL queries of all complexities. The question bank allows the teachers to create test sets with the questions it contains and to assign the test sets to individual students. SQL-LES also supports management of student projects. The system supports automatic evaluation of SQL expressions to a limited extent.

Soler et al [11] proposes a web-based SQL learning tool for automatic correction of SQL statements. The authors named the tool SQL-ACME which is integrated in an e-learning framework, ACME. In the SQL-ACME tool, a student enters a solution i.e. an SQL expression for a particular problem. There is a correct SQL statement for the problem stored in the system database. The result set returned by the student's SQL statements is matched with the result set returned by the correct query statement. If they match the system shows the result of the evaluation as correct. If the result sets do not match then the system shows incorrect as the result of the evaluation. If a result set of a submitted SQL statement is in different order it can still get correct as the evaluation result as long as the result set has all the results that the correct result set contains. For an example,



if a submitted SQL expression returns result set in the order of name, id instead of id, name the statement can be marked as correct if the result set returned matches with the correct one.

Sadiq et al [12] presents SQLator, a web-based tool for learning SQL. Key features of the system include multimedia tutorials, personalized learning, interaction with teachers, a query execution environment, different databases with practice queries and evaluation of practice queries. A learner starts with selecting a database and then selecting a practice query problem and writing an SQL statement to solve the problem. One of the key features of SQLator is the evaluator which evaluates the SQL statement answered by the learner as either correct or incorrect.

## 2.3 Similarity between Nominal Attributes

There are a few techniques for measuring similarity and dissimilarity between two objects [13]. A technique for measuring similarity or dissimilarity is chosen based on the type of attributes the objects have. In this section we will present the similarity measure for two objects having nominal attributes [13]. In our partial evaluation model, we present some of the equations in a format very similar to the equation used for measuring similarity of two objects with nominal attributes.

Let us imagine a matrix containing  $N$  rows and  $M$  columns. Each row of this matrix represents an object and each column represents an attribute which implies that we have  $N$  objects with each object having  $M$  attributes. We can refer to row 1 as object 1, row 2 as object 2 and similarly row  $N$  as object  $N$ . Now the similarity between objects  $i$  and  $j$  is calculated using the equation presented below.

$$sim(i, j) = \frac{m}{p}$$

In the above equation  $m$  is the number of attributes for which  $i$  and  $j$  are in the same state and  $p$  is the total number of attributes describing the objects.

# Chapter 3

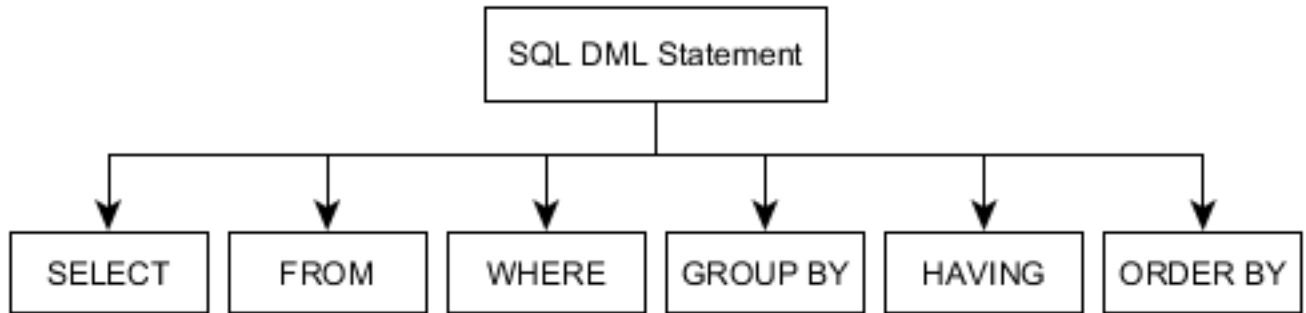
## Partial Evaluation Model

In this chapter we describe our proposed model for partial evaluation of SQL expressions. Our model assigns partial score to an answered SQL expression that is partially correct. The scoring is achieved by comparing the answered query expression with a correct query expression for the respective question. We call the correct expression as the reference expression. The score of the answered expression is high if the similarity between the answered expression and the reference expression is high and it is vice versa. In the following section we discuss the different SQL clauses and an overview of our partial evaluation model.

### 3.1 Analysis of SQL Statement

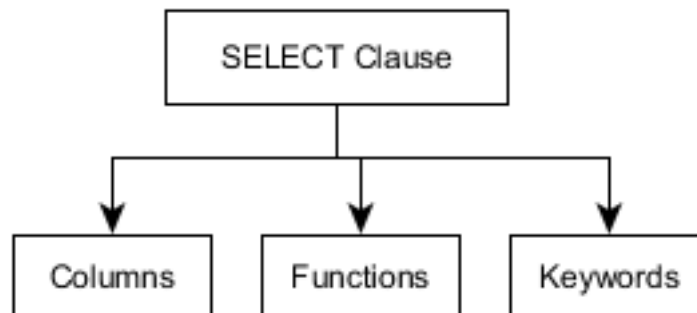
As mentioned earlier, an answered SQL expression is scored by comparing it with the reference expression and measured how similar the answered expression is to the reference expression. We have applied a top-down approach to solve the problem of measuring similarity by dividing the expression into smaller sub-expressions followed by solving the sub-problems and combining the sub-solutions to form the final solution.

An SQL DML expression can contain at most six clauses (Figure 3.1): SELECT, FROM, WHERE, GROUP BY, HAVING and ORDER BY. To measure a similarity score in our model, an SQL expression is first broken into the individual clauses. Then an individual score is measured for each of the clauses separately. The basis for calculation of the individual scores is similarity. Both the answered and the reference expressions are broken into individual clauses. Then individual scores for the clauses are calculated by measuring the similarity of the clauses in the answered expression with the respective clauses in the reference expression. The individual scores are finally combined to calculate the final score for the answered expression.



**Figure 3.1:** Structure of an SQL DML Expression

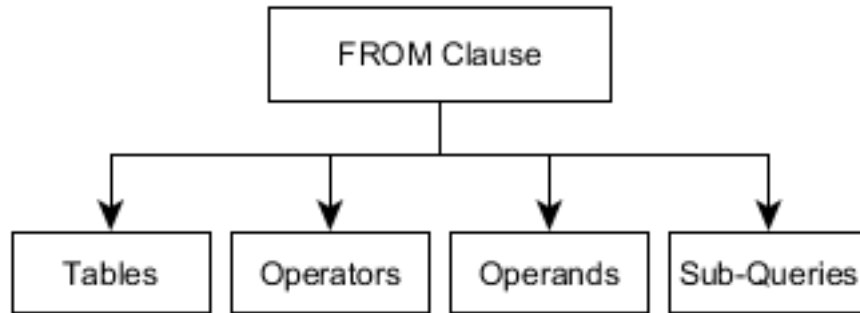
To find the score of individual clause we consider the elements that the clause contains. Figure 3.2 shows the individual elements that are contained within a SELECT clause. A SELECT clause may contain columns (or fields), functions and/or keywords. The answered query expression's SELECT may contain the elements matched with the reference expression's SELECT. It may also contain elements unmatched to the reference SELECT. Both matched and unmatched elements (i.e. fields, functions and/or operator) in the answer's SELECT and the elements contained in the reference's SELECT are the key elements used to score the SELECT clause.



**Figure 3.2:** Structure of SQL SELECT Clause

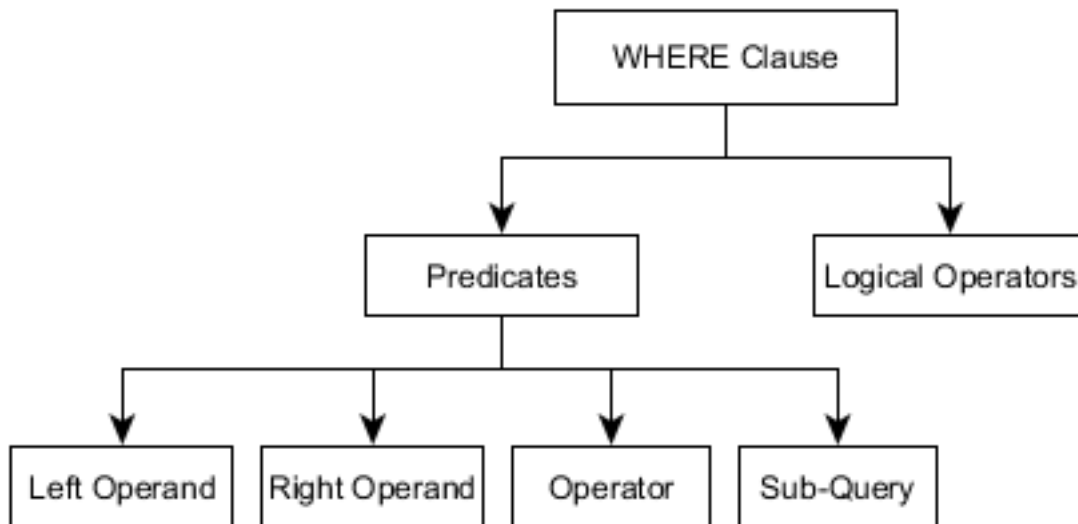
A FROM clause (Figure 3.3) may contain tables, operators, operands and in some cases sub-queries. While scoring the FROM clause, we consider matched and unmatched elements (i.e. tables, operators, operands and/or sub-queries) in the answer's FROM clause and the elements that reference expression's FROM clause contains.

A WHERE clause (Figure 3.4) contains one or more predicates joined by logical operators. A predicate in turn contains operators, an operand and sometimes sub-query. The predicates are



**Figure 3.3:** Structure of SQL FROM Clause

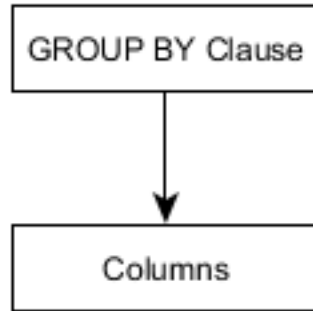
scored individually using the matched and unmatched operators, operands and/or sub-queries in the answered expression and those elements contained in the reference expression. The scores for the conditions are combined along with the score for the logical operators to compute the score of WHERE clause.



**Figure 3.4:** Structure of SQL WHERE Clause

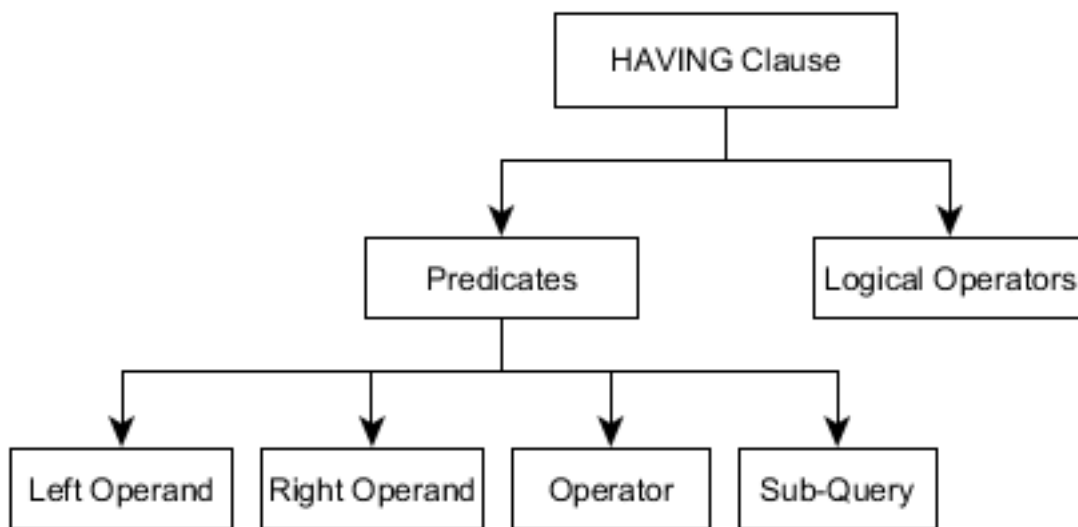
Only fields/columns are considered to score a GROUP BY clause (Figure 3.5).

A HAVING clause (Figure 3.6) is very similar to the WHERE clause and contains similar elements to the latter. Like a WHERE clause, a HAVING clause may contain one or more predicates and logical operators. Also like WHERE, Each predicate in HAVING may contain operators, operand and/or sub-query. One significant difference is that a predicate in the HAVING clause



**Figure 3.5:** Structure of SQL GROUP BY Clause

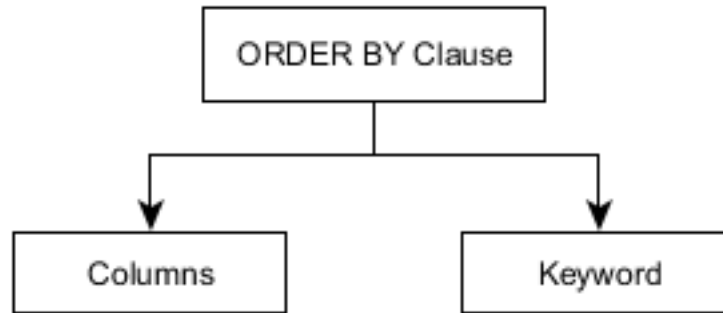
may contain an aggregate function which can never be the case in WHERE. Due to the significant similarity that WHERE and HAVING share, the scoring of HAVING is also very similar to the scoring of WHERE.



**Figure 3.6:** Structure of SQL HAVING Clause

An ORDER BY clause (Figure 3.7) may contain columns/fields and operators/keywords (such as ASC, DESC). To score the ORDER BY clause we take the answered query's matched and unmatched elements (columns and operator) and the reference query's total elements in the ORDER BY into consideration.

To calculate the final score, the individual scores are combined in such a way that resembles the clauses according to their complexities. In simple words, a clause with more complexity should



**Figure 3.7:** Structure of SQL ORDER BY Clause

carry more weight in the final score. Even though an SQL expression may contain at most six clauses all the clauses may not be present in an expression. Furthermore, it may occur that while the reference expression contains a particular clause, the clause might be absent from the answered expression. An opposite scenario may also occur.

A clause might be present in both the answered query expression and the reference query expression, or a clause might be present only in the reference expression, or it might be present only in the answered expression. If the clause is present in both the answered and reference expressions a score will be calculated for the clause by measuring the similarity between the clauses present in both the answered and reference expressions. If the clause is present only in the reference expression (and absent in the answered expression) then a score of zero will be assigned for that clause. If a clause is present only in the answered query expression then a negative score will be assigned to it. Finally if a clause is absent in both the answered and the reference expressions then it will not be scored and will not be considered while calculating the final score.

For an example, let an answered query expression has three clauses SELECT, FROM and WHERE. The reference query expression has four clauses SELECT, FROM, WHERE and ORDER BY. For simplicity, let's assume that all the SELECT, FROM and WHERE clauses in the answered expression are exactly similar to the SELECT, FROM and WHERE clauses in the reference expression. Now as there is no ORDER BY in the answered expression, it will get a zero in the ORDER BY clause. But the answered expression will get full marks in the SELECT, FROM and WHERE clauses as they are exactly same to those clauses in the reference expression and zero in ORDER BY clause. The final score will be a weighted mean of all the individual scores.

Let's consider a second scenario in which an answered query expression has SELECT, FROM and WHERE clauses and the reference query expression has SELECT, FROM, WHERE and ORDER BY clauses. The SELECT and FROM clauses in the answered query expression are exactly similar to those clauses in the reference query expression but the WHERE clause is not exactly similar to the reference expression's WHERE clause. Let's assume the WHERE clause in the reference expression has three conditions with two logical operators whereas the answered expression's WHERE clause has two conditions that are exactly same to two of the reference expression's conditions and one logical operator which is same to the logical operators between those two conditions in the reference expression. Now as the WHERE clause is not exactly same to that of the reference query, the answered query will get a partial score (instead of a full score or a zero) in the WHERE clause. The answered query gets full scores for SELECT and FROM, partial score for WHERE and zero for ORDER BY. So the answered query will be assigned a lesser score than the previous one.

Syntax error is an important point to keep in mind while scoring an answer. In our model, the scoring of an answered expression goes on until a syntax error occurs. At the moment, a syntax error occurs, the scoring is stopped. It should be noted that most likely a partial score has already been calculated before stopping the scoring due to a syntax error. In that case the final score at that point should be the score calculated so far. Also if an answered expression contains only a SELECT clause then a zero shall be assigned to that answer.

## 3.2 Scoring SELECT Clause

As discussed already, each of the clauses of the answered expression are scored individually with respect to the reference query expression. This section presents the scoring technique used for the SELECT clause. The equation that assigns the score to the SELECT clause is presented below:

$$S_{sl} = \frac{m - u/t}{p} \quad (3.1)$$

$S_{sl}$ : Similarity score for SELECT

$m$ : Number of elements i.e. columns(or fields), functions and/or keywords in the answered expression's SELECT clause matched with the elements in the reference expression's SELECT clause

$u$ : Number of unmatched elements in the answered expression's SELECT clause

$p$ : Number of columns, functions and/or keywords in reference expression's SELECT clause

$t$ : How much of  $u$  should be considered as penalty (if  $t$  is 1 the whole  $u$  will be considered as penalty and greater the  $t$ , lesser the penalty;  $t$  cannot be zero)

In case the SELECT clauses in both the queries are perfect match the score will be 1 as  $m$  will be equal to  $p$  and  $u$  will be 0.

When there are no matched fields and functions and/or keyword and one or more unmatched fields, functions, keywords in the answered expression's SELECT clause the score will be negative. It should be noted that no negative score is allowed for any of the clauses unless the clause is an extra clause. The lowest possible score for SELECT (and other clauses) is 0. So if the score calculated is lesser than 0, 0 is reassigned to  $S_{sl}$ .

### 3.3 Scoring FROM Clause

The FROM clause may contain name of tables with or without JOIN operations and/or sub-queries.

This results in three possible scenarios:

1. a FROM clause with only tables with/without JOIN operations,
2. a FROM clause with only sub-queries,
3. a FROM clause where both the tables with/without JOIN and sub-queries are present.

Due to these, to score a FROM clause, two other scores (in case FROM clause contains both tables and sub-query) might be necessary to be calculated:  $S_{fr0}$  for the tables with/without JOIN and



$S_{fr1}$  for sub-queries. The expression for  $S_{fr0}$  is as follows.

$$S_{fr0} = \frac{m - u/t}{p} \quad (3.2)$$

$m$ : Number of matched elements (i.e. tables, operators and operands) in the answered expression's FROM clause

$u$ : Number of unmatched elements in the answered expression's FROM clause

$p$ : Number of elements in reference expression's FROM clause

$t$ : How much of  $u$  should be considered as penalty (if  $t$  is 1 the whole of  $u$  will be considered as penalty and greater the  $t$  lesser the penalty;  $t$  cannot be zero)

It should be noted that no negative score is allowed for  $S_{fr0}$ . If  $S_{fr0}$  calculated is lesser than 0, 0 should be reassigned.

As a sub-query is essentially an individual SQL query score  $S_{fr1}$  is calculated recursively using the very same technique that is used for determining the overall score of an answered query expression.

Now when the FROM clause contains only the tables with/without JOIN, the score,  $S_{fr}$  assigned to the FROM clause is  $S_{fr0}$ . When the FROM clause contains only sub-queries,  $S_{fr}$  assigned to the FROM clause is  $S_{fr1}$ . If the FROM clause contains both the tables and sub-queries then  $S_{fr}$  is a weighted average of  $S_{fr0}$  and  $S_{fr1}$  with greater weights assigned to  $S_{fr1}$ .

So, for FROM clause with only tables:

$$S_{fr} = S_{fr0}$$

For FROM clause with only sub-queries:

$$S_{fr} = S_{fr1}$$

For FROM clause with both tables and sub-queries:

$$S_{fr} = \frac{w_{fr0}S_{fr0} + w_{fr1}S_{fr1}}{w_{fr0} + w_{fr1}} \quad (3.3)$$

In the above equation,  $w_{fr0}$  and  $w_{fr1}$  are weights for  $S_{fr0}$  and  $S_{fr1}$  respectively.

### 3.4 Scoring WHERE Clause

To obtain the score for the WHERE clause some other scores,  $S_{wpr}$  and/or  $S_{wpr\_sb}$  and  $S_{wlp}$  are required.  $S_{wpr}$  and  $S_{wpr\_sb}$  are the scores for the predicates without sub-queries and the predicates with sub-queries respectively in the WHERE clause. To obtain  $S_{wpr}/S_{wpr\_sb}$  we have to find a score for every individual predicate ( $S_{wpr\_i}/S_{wpr\_sb\_i}$ ).  $S_{wlp}$  is the score for the logical operators in the WHERE clause.

A predicate within a WHERE clause may or may not contain sub-queries. Due to this, predicates that contain sub-queries are scored using a different function than the predicates that have no sub-queries. When there are no sub-queries, the score for the  $i$ th predicate  $S_{wpr\_i}$  is obtained using the formula below:

$$S_{wpr\_i} = \frac{x_{ld\_i} + x_{rd\_i} + x_{op\_i}}{w_{opd}} \quad (3.4)$$

$x_{ld\_i}$ ,  $x_{rd\_i}$  and  $x_{op\_i}$ : Discrete variables for left operand, right operand and operator respectively.

$$x_{ld\_i} \in \{u_1, u_2\}$$

$x_{ld\_i}$  is  $u_1$  if the left operands in the answered and reference expressions match.  $x_{ld\_i}$  is  $u_2$  if the left operands do not match. If the answered query expressions match then the answered query will be rewarded so  $u_1$  is a positive real number. On the other hand if they do not match then the answered query will be penalized so  $u_2$  is a negative real number.

$$x_{rd\_i} \in \{v_1, v_2, v_3\}$$

$x_{rd\_i}$  is  $v_1$  if the right operands in the answered and reference expressions match.  $x_{rd\_i}$  is  $v_2$  if the right operands do not match. The answered expression will be rewarded with  $v_1$  assigned to  $x_{rd\_i}$  if the right operands match and it will be penalized with  $v_2$  if the right operands do not match. So  $v_1$  is positive and  $v_2$  is negative. In some cases, the right operand in the answered expression may get close to the right operand in the reference expression but they may not be an exact match. For an example if the right operand in the reference query is '%programming%' whereas in the answered query the right operand is 'Programming' the answered query's right operand neither will be penalized with  $v_2$  nor it will be rewarded with  $v_1$ . Instead,  $x_{rd\_i}$  will take another value  $v_3$  such as  $v_2 < v_3 < v_1$ .

$x_{op\_i} \in \{k_1, k_2\}$  [ $k_1$  if operators match,  $k_1 \geq 0$ ;  $k_2$  otherwise;  $k_2$  negative].

$w_{opd}$ : Sum of the highest possible values for  $x_{ld\_i}$ ,  $x_{rd\_i}$  and  $x_{op\_i}$ .

$$w_{opd} = u_1 + v_1 + k_1 \quad (3.5)$$

Here  $u_1, u_2, v_1, v_2, v_3, k_1, k_2$  are real numbers.

$S_{wpr\_i}$  must not be negative. 0 is assigned to  $S_{wpr\_i}$  if it becomes negative. There is an exception to this if the predicate in question is an extra predicate. If the number of predicates in the answered query's WHERE clause that have already been scored is equal to the number of predicates in the reference query's WHERE clause then the remaining predicates in the answered query's WHERE clause are considered extra predicates. There is no need to score an extra predicate using the respective scoring function. An extra predicate is directly assigned a negative score.

$S_{wpr\_i}$  gives us the score for an individual predicate (i.e.  $i$ th predicate). If a WHERE clause has  $n$  predicates we shall have  $n$  such scores. We are yet to calculate the overall score for the predicates  $S_{wpr}$  which is presented below:

$$S_{wpr} = \frac{\sum_{i=1}^n S_{wpr\_i}}{N} \quad (3.6)$$

In the above function,  $n$  is the number of predicates that the answered expression's WHERE clause contain and  $N$  is the number of predicates that are present in the reference expression's WHERE clause.  $S_{wpr\_i}$  is the score of  $i$ th predicate.

When there is a sub-query within a predicate then the score  $S_{wpr\_sb\_i}$  will be:

$$S_{wpr\_sb\_i} = \frac{x_{ld\_i} + w_{rd\_i}S_{sb\_i} + x_{op\_i}}{w_{opd}} \quad (3.7)$$

$x_{op\_i}$  and  $x_{ld\_i}$ : Discrete variables for operator and left operand respectively.

$w_{rd\_i}$ : Weight for right operand.

$x_{ld\_i} \in \{u_1, u_2\}$  [ $u_1$  if left operand matches,  $u_1$  positive;  $u_2$  otherwise;  $u_2$  negative].

$x_{op\_i} \in \{k_1, k_2\}$  [ $k_1$  if operator matches,  $k_1 \geq 0$ ;  $k_2$  otherwise;  $k_2$  negative].

$w_{opd}$ : Sum of  $w_{rd\_i}$  and the highest possible values for  $x_{ld\_i}$  and  $x_{op\_i}$ .

$$w_{opd} = u_1 + w_{rd\_i} + k_1 \quad (3.8)$$

$S_{sb\_i}$ : Score for the sub-query (out of 1) calculated recursively.

If  $S_{pr\_sb\_i}$  is negative then 0 is reassigned to it.

The overall score for predicates with sub-queries is as follows:

$$S_{wpr\_sb} = \frac{\sum_{i=1}^n S_{wpr\_sb\_i}}{N} \quad (3.9)$$

Here  $n$  is the number of predicates with sub-queries that the answered query's WHERE clause contain and  $N$  is the number of predicates with sub-queries that are present in the reference query's WHERE clause.  $S_{wpr\_sb\_i}$  is the score of  $i$ th predicate.

Scoring predicates based on similarity can be challenging as predicates can be in any order. For an example, the first predicate in the answered expression may be very similar to the second predicate in the reference query. In that case scoring the first predicate in the answered expression with respect to the first predicate in the reference query expression may result in a poor score (or zero) but for the second predicate in the reference query the answered query predicate may obtain a good score. Choosing the predicate right in the reference query is crucial to accurately score the predicates. A solution to this problem is with respect to each predicate in the reference query expression all the predicates that have not been assigned a score yet in the answered query is scored and the predicate for which the score is maximum is considered scored. The steps for calculating scores for the predicates is presented below:

1. Select the first predicate in the reference query
2. Consider the currently selected predicate as the current reference predicate
3. Calculate a score for every available predicate in the answered query with respect to the current reference predicate
4. Select the predicate in the answered query for which the score is the maximum and consider the predicate scored and discard the predicate from the list of available predicates
5. Now select the next predicate in the reference query and repeat steps 2-4 until there is no predicate left in the reference query or until there is no predicate left in the answered query
6. Consider the remaining predicates in the answered query as extra predicates and assign them appropriate negative scores

Calculation for the score for logical operators,  $S_{wlp}$  is presented below:

$$S_{wlp} = \frac{m - u/t}{p} \quad (3.10)$$

$m$ : Number of matched logical operators in the answered expression's WHERE clause.

$u$ : Number of unmatched logical operators in the answered expression's WHERE clause.

$t$ : How much of  $u$  to be considered for penalty.

$p$ : Number of logical operators in the reference query expression.

If there are no sub-queries then the weights for the predicates and logical operators are chosen carefully such that the predicates have much greater weight than the logical operators. But if there are predicates with sub-queries then weight assignment is not as straight forward as in the former scenario. Generally the idea is to assign greater weight to the predicates with sub-queries and lesser weight to the predicates that contain no sub-queries. But we need to know the ratio of the number of predicates with sub-queries to the number of predicates without sub-queries to assign weights that are consistent with the actual scenario. In other words, if there are predicates with sub-queries, then we need to calculate the weights before assigning them.

In case there are sub-queries in the WHERE clause, the weights for predicates without sub-queries and predicates with sub-queries as well as the weight for the logical operators are calculated using the following equations:

$$w_{wpr} = \frac{N_{pr}}{N} \quad (3.11)$$

$$w_{wpr\_sb} = c \frac{N_{pr\_sb}}{N} \quad (3.12)$$

$$w_{wlp} = \frac{1}{2} \text{Min}(w_{wpr}, w_{wpr\_sb}) \quad (3.13)$$

$w_{wpr}$ ,  $w_{wpr\_sb}$  and  $w_{wlp}$  are weights for predicate without sub-queries, predicates with sub-queries and logical operators respectively.

$N_{pr}$ : Number of predicates without sub-queries in the reference query.

$N_{pr\_sb}$ : Number of predicates with sub-queries in the reference query.

$N$ : Total number of predicates in the reference query i.e.  $N = N_{pr} + N_{pr\_sb}$

$c$ : A factor that adds more importance to the predicates with sub-queries.

Generally the weight for predicates with sub-queries should be larger than the weight for predicates without sub-queries. If the number of predicates with sub-queries and the number of predicates without sub-queries are equal i.e.  $N_{pr} = N_{pr\_sb}$  the weight for predicates with sub-queries must be larger. If the number of predicates without sub-queries ( $N_{pr}$ ) is slightly greater than the number predicates with sub-queries ( $N_{pr\_sb}$ ) the weight for the predicates with sub-queries  $w_{wpr\_sb}$  should still be larger than or equal to or atleast closer (depending on how much greater  $N_{pr}$  is) to the weight for predicates without sub-queries ( $w_{wpr}$ ). However, if  $N_{pr}$  is significantly greater than  $N_{pr\_sb}$  then  $w_{wpr}$  must be greater than  $w_{wpr\_sb}$ . To ensure this  $c$  must be greater than 1 but  $c$  should not be too large.

Now the score for the WHERE clause,  $S_{wh}$  is calculated from the following function:

$$S_{wh} = \frac{w_{wpr}S_{wpr} + w_{wpr\_sb}S_{wpr\_sb} + w_{wlp}S_{wlp}}{w_{wpr} + w_{wpr\_sb} + w_{wlp}} \quad (3.14)$$

$S_{wpr}$ ,  $S_{wpr\_sb}$  and  $S_{wlp}$ : Scores for predicates without sub-queries, predicates with sub-queries and logical operators respectively.

$w_{wpr}$ ,  $w_{wpr\_sb}$  and  $w_{wlp}$ : Weights for predicates without sub-queries, predicates with sub-queries and logical operators respectively.

### 3.5 Scoring GROUP BY Clause

Scoring for Group By clause is very similar to that of SELECT and FROM. For GROUP BY clause, the function for calculating the score is presented below:

$$S_{gb} = \frac{m - u/t}{p} \quad (3.15)$$

$S_{gb}$ : Similarity score for GROUP BY clause

$m$ : Number of matched columns in answered expression's GROUP BY clause

$u$ : Number of unmatched columns in the answered expression's GROUP BY clause

$p$ : Number of columns in reference query's GROUP BY clause

$t$ : How much of  $u$  should be considered as penalty;  $t$  cannot be zero

No negative score is allowed for  $S_{gb}$ . In case  $S_{gb}$  becomes negative, 0 will be reassigned.

### 3.6 Scoring HAVING Clause

Scoring of HAVING clause is very similar to the scoring of WHERE clause as they are very similar.

Like the WHERE clause, scoring the HAVING clause requires scores for the predicates without sub-queries and predicates with sub-queries,  $S_{hpr}$ ,  $S_{hpr\_sb}$  and a score for the logical operators,  $S_{hlp}$ . To find  $S_{hpr}/S_{hpr\_sb}$ , the scores for the individual predicates,  $S_{hpr\_i}/S_{hpr\_sb\_i}$  must be calculated.

A predicate within a HAVING clause may or may not contain sub-queries. So predicates that contain sub-queries are scored differently than the predicates with no sub-queries. When there are no sub-queries,  $S_{hpr\_i}$  is obtained using the below function:

$$S_{hpr\_i} = \frac{x_{ld\_i} + x_{rd\_i} + x_{op\_i}}{w_{opd}} \quad (3.16)$$

$x_{ld\_i}$ ,  $x_{rd\_i}$  and  $x_{op\_i}$ : Discrete variables for left operand, right operand and operator respectively.

$x_{ld\_i} \in \{u_1, u_2\}$  [ $u_1$  if left operand matches,  $u_1$  positive;  $u_2$  otherwise;  $u_2$  negative].

$x_{rd\_i} \in \{v_1, v_2, v_3\}$  [ $v_1$  if right operands match,  $v_1$  positive;  $v_2$  if right operands do not match,  $v_2$  negative;  $v_3$  if the operands do not perfectly match but they somewhat match,  $v_2 < v_3 < v_1$ ].

$x_{op\_i} \in \{k_1, k_2\}$  [ $k_1$  if operator matches  $k_1 \geq 0$ ;  $k_2$  otherwise;  $k_2$  negative].



$w_{opd}$ : Sum of the highest possible values for  $x_{ld-i}$ ,  $x_{rd-i}$  and  $x_{op-i}$ .

$$w_{opd} = u_1 + v_1 + k_1 \quad (3.17)$$

Here  $u_1, u_2, v_1, v_2, v_3, k_1, k_2$  are all real numbers.

$S_{hpr-i}$  must not be negative. So the lowest possible value for  $S_{hpr-i}$  is 0. 0 is reassigned to  $S_{hpr-i}$  if it becomes negative.

We are yet to calculate the overall score for the predicates  $S_{hpr}$  which is presented below:

$$S_{hpr} = \frac{\sum_{i=1}^n S_{hpr-i}}{N} \quad (3.18)$$

In the above equation, n is the number of predicates that the answered query's HAVING clause contain and N is the number of predicates that are present in the reference query's HAVING clause.  $S_{hpr-i}$  is the score of ith predicate.

When there is a sub-query within a predicate then the score  $S_{hpr-sb-i}$  will be:

$$S_{hpr-sb-i} = \frac{x_{ld-i} + w_{rd-i}S_{sb-i} + x_{op-i}}{w_{opd}} \quad (3.19)$$

$x_{ld-i}$  and  $x_{op-i}$ : Discrete variables for left operand and operator respectively.

$w_{rd-i}$ : Weight for right operand which is a sub-query.

$S_{sb-i}$ : Score for the sub-query (out of 1) calculated recursively.

$x_{ld-i} \in \{u_1, u_2\}$  [ $u_1$  if left operand matches,  $u_1$  positive;  $u_2$  otherwise;  $u_2$  negative].

$x_{op-i} \in \{k_1, k_2\}$  [ $k_1$  if operator matches,  $k_1 \geq 0$ ;  $k_2$  otherwise;  $k_2$  negative].

$w_{opd}$ : Sum of  $w_{rd-i}$  and the highest possible values for  $x_{ld-i}$  and  $x_{op-i}$ .

$$w_{opd} = u_1 + w_{rd_i} + k_1$$

If  $S_{hpr\_sb\_i}$  is negative then 0 is reassigned to it.

The overall score for predicates with sub-queries is as follows:

$$S_{hpr\_sb} = \frac{\sum_{i=1}^n S_{hpr\_sb\_i}}{N} \quad (3.20)$$

Here  $n$  is the number of predicates with sub-queries that the answered query's HAVING clause contains and  $N$  is the number of predicates with sub-queries that are present in the reference query's HAVING clause.  $S_{hpr\_sb\_i}$  is the score of  $i$ th predicate.

As discussed in section 3.4 scoring the predicates in the correct order can be challenging. We already presented the solution to this problem in the said section. The same solution is applicable for scoring predicates in the HAVING clause. If there are extra predicates in the answered expression, each of the extra predicates will be assigned a negative score.

Calculation for the score for logical operators,  $S_{hlp}$  is presented below:

$$S_{hlp} = \frac{m - u/t}{p} \quad (3.21)$$

$m$ : Number of matched logical operators in the answered expression's HAVING clause.

$u$ : Number of unmatched logical operators in the answered expression's HAVING clause.

$t$ : How much of  $u$  to be considered for penalty.

$p$ : Number of logical operators in the reference query expression.

If there are sub-queries in the HAVING clause then the weights for the predicates with sub-queries, predicates without sub-queries and the logical operators are calculated. The calculation

for these weights are presented below:

$$w_{hpr} = \frac{N_{pr}}{N} \quad (3.22)$$

$$w_{hpr\_sb} = c \frac{N_{pr\_sb}}{N} \quad (3.23)$$

$$w_{hlp} = \frac{1}{2} \text{Min}(w_{hpr}, w_{hpr\_sb}) \quad (3.24)$$

$w_{hpr}$ ,  $w_{hpr\_sb}$  and  $w_{hlp}$ : Weights for predicates without sub-queries, predicates with sub-queries and logical operators respectively.

$N_{pr}$ : Number of predicates without sub-queries in the reference SQL expression.

$N_{pr\_sb}$ : Number of predicates with sub-queries in the reference SQL expression.

$N$ : Total number of predicates in the reference expression i.e.  $N = N_{pr} + N_{pr\_sb}$

$c$ : Factor that adds more importance to the predicates with sub-queries.  $c > 1$ .  $c$  should not be too large.

Now, the score for the HAVING clause,  $S_{hv}$  is calculated from the following function:

$$S_{hv} = \frac{w_{hpr}S_{hpr} + w_{hpr\_sb}S_{hpr\_sb} + w_{hlp}S_{hlp}}{w_{hpr} + w_{hpr\_sb} + w_{hlp}} \quad (3.25)$$

$S_{hpr}$ ,  $S_{hpr\_sb}$  and  $S_{hlp}$ : Scores for predicates without sub-queries, predicates with sub-queries and logical operators.

### 3.7 Scoring ORDER BY Clause

For ORDER BY clause, the score is obtained from the same function used for SELECT, FROM and GROUP BY clause:

$$S_{ob} = \frac{m - u/t}{p} \quad (3.26)$$

$S_{ob}$ : Similarity score for ORDER BY clause

$m$ : Number of elements (columns, operator) in the answered expression's ORDER BY clause matched with the elements in the reference expression's ORDER BY clause

$u$ : Number of unmatched elements in the answered query expression

$p$ : Number of elements in reference expression's ORDER BY clause

$t$ : How much of  $u$  should be considered as penalty;  $t$  cannot be zero

### 3.8 Overall Score

The scores obtained for all the clauses present in the reference query discussed in the previous sections are finally combined to obtain the final overall score for the answered query. The final score is obtained from the following function:

$$S_f = \frac{W_{sl}S_{sl} + W_{fr}S_{fr} + W_{wh}S_{wh} + W_{gb}S_{gb} + W_{hv}S_{hv} + W_{ob}S_{ob}}{W_{sl} + W_{fr} + W_{wh} + W_{gb} + W_{hv} + W_{ob}}M \quad (3.27)$$

In the above equation,  $W_{sl}$  is the weight assigned for SELECT clause and  $S_{sl}$  is the score calculated for the SELECT clause. Similarly,  $W_{fr}$ ,  $W_{wh}$ ,  $W_{hv}$ ,  $W_{gb}$  and  $W_{ob}$  are weights and  $S_{fr}$ ,  $S_{wh}$ ,  $S_{hv}$ ,  $S_{gb}$  and  $S_{ob}$  are scores for FROM, WHERE, HAVING, GROUP BY and ORDER BY clauses respectively.  $M$  is the mark that a given question carries.

### 3.9 Scoring Multiple Queries

Some SQL queries may have multiple queries joined by set operators such as union, minus and intersect. In the case of those queries, scores are calculated for the sub-queries separately.

The following function gives us the final score of the entire query:

$$S_f = \frac{S_{f1} + S_{f2}}{2}x_{sp} \quad (3.28)$$

$S_{f1}$  and  $S_{f1}$  are the scores for the first query and the second query.

$$x_{sp} \in \{u, v\}$$

$x_{sp} = u$  if set operators match;  $v$  if set operators do not match.

# Chapter 4

## Experimental Results

In this chapter, we evaluate our model by applying it to score various query expressions with respect to respective reference expressions. We have collected scores from experts for a set of SQL query expressions to make comparison. Our experiments show promising results.

### 4.1 Evaluation Methodology

We have prepared four data sets that contain SQL expressions of various complexities. The data sets have been presented in Appendices. Table 4.1 shows different types of queries we considered to test our model. We consider 7 types of queries. For a query to be labeled as a certain TYPE the query must contain the components required for the TYPE. Additionally it may (or may not) contain the optional components for that TYPE. Table 4.2 presents a summary of the four data sets. Two of the data sets (2nd and 4th data sets ) contain real world data that have been part of SQL-LES, an e-learning system used in a database course.

We have approached three experts to score the SQL answers in our data sets. We have applied our model to score the same answers in our data sets. Finally we have analyzed the performance of our model by comparing the scores generated from our model and the scores assigned by experts. We show the results of comparison in tables and charts. We categorize the scores generated by our model in different similarity categories based on how similar they are with respect to the scores assigned by human experts. A score is considered Extremely Similar for an answer if the difference between the score by the expert and the score by the model is within 5 percent. A score is Very Similar if the difference is within 10 percent. A score is considered Extremely Dissimilar if the difference is more than 40 percent. We have created 6 such categories. Table 4.3 shows the categories and their criteria. We will use these categories to analyze our result throughout this

**Table 4.1:** Types of SQL Query Expressions

| Type   | Required Components   | Optional Components                             |
|--------|---|---|
| TYPE-1 | SELECT, FROM  | Functions in SELECT                             |
| TYPE-2 | SELECT, FROM, WHERE with at most 2 predicates                   | ORDER BY, Functions in SELECT                   |
| TYPE-3 | SELECT, FROM, WHERE with at least 3 predicates                  | ORDER BY, Functions in SELECT                   |
| TYPE-4 | SELECT, FROM, WHERE, GROUP BY                                   | ORDER BY, Functions in SELECT                   |
| TYPE-5 | SELECT, FROM, WHERE, GROUP BY, HAVING                           | ORDER BY, Functions in SELECT                   |
| TYPE-6 | SELECT, FROM, WHERE, Sub-query in WHERE                         | GROUP BY, HAVING, ORDER BY, Functions in SELECT |
| TYPE-7 | Multiple queries with set operator (union, minus, intersection) | Any other components                            |

**Table 4.2:** Summary of Data

| Data Set   | Number of SQL Questions | Number of SQL Answers | Types of Queries Present               |
|------------|-------------------------|-----------------------|--|
| Data-Set 1 | 7                       | 12                    | TYPE-2, TYPE-3                         |
| Data-Set 2 | 9                       | 16                    | TYPE-2, TYPE-3, TYPE-4, TYPE-6, TYPE-7 |
| Data-Set 3 | 8                       | 10                    | TYPE-1, TYPE-3, TYPE-4, TYPE-5         |
| Data-Set 4 | 8                       | 12                    | TYPE-6, TYPE-7                         |

chapter.

For experimentation we calculated scores by counting the number of matched/unmatched elements, determining values for left/right operands and operators by checking if they match or not. We prepared the numerical data set manually for all matched and unmatched elements. We wrote python program to implement the equations in the model. The program takes the manually determined values as parameters and returns a score.

We present results for each data set individually. For each data set we first present the comparison of scores assigned by experts and our model in a clustered bar-chart. We then count the similar and dissimilar scores based on the categories presented in Table 4.3. We show these results as frequency tables and bar-charts. Table 4.4 shows different values for weights and variables

**Table 4.3: Similarity Categories**

| Categories           | Score Difference between model and expert      |
|----------------------|--|
| Extremely Similar    | Score Difference $\leq  5\% $                  |
| Very Similar         | $ 5\%  < \text{Score Difference} \leq  10\% $  |
| Similar              | $ 10\%  < \text{Score Difference} \leq  20\% $ |
| Dissimilar           | $ 20\%  < \text{Score Difference} \leq  30\% $ |
| Very Dissimilar      | $ 30\%  < \text{Score Difference} \leq  40\% $ |
| Extremely Dissimilar | Score Difference $>  40\% $                    |

present in the model that we use to calculate our score.

While evaluating our model we had an interesting observation. As data-set 2 was evaluated by two experts separately, the scores by the experts for the same answer varied significantly in some cases. This reminded us of a human factor that is involved in scoring. As there is no single guideline in scoring/grading and the scoring policy differs from one scorer to the next scoring in most cases is a subjective task.

## 4.2 Model Evaluation

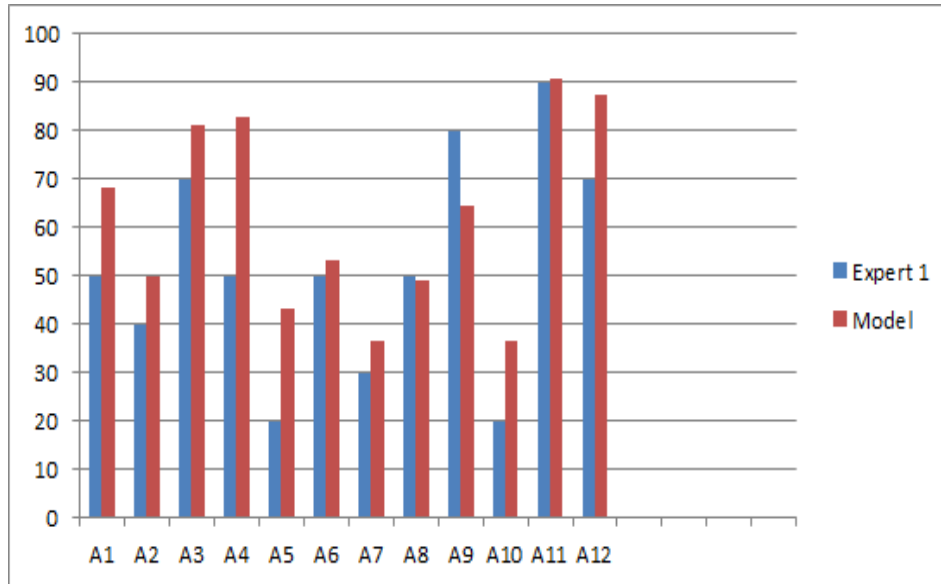
In this section, we present the result of our experimentation with four data sets. We first present our evaluation with data-set 1. We collected evaluation from 1st expert for the first data set. We show the comparison between the scores assigned by the 1st expert and our model for the 1st data set in Figure 4.1. We show the similarity and dissimilarity counts for the 1st data set as a frequency table in Table 4.5 and as a bar-chart in Figure 4.2.

As the result shows in 25% of the cases scores generated by our model is extremely similar to the scores assigned by the first expert for data-set 1. If we create just two categories 'similar' and 'dissimilar' and if similar includes the categories Extremely Similar, Very Similar and Similar while dissimilar includes the categories Dissimilar, Very Dissimilar and Extremely Dissimilar then for 83.34% of the answers the scores generated by our model is similar to the scores assigned by the 1st expert for data-set 1.



**Table 4.4:** Values used for different variables in partial evaluation model

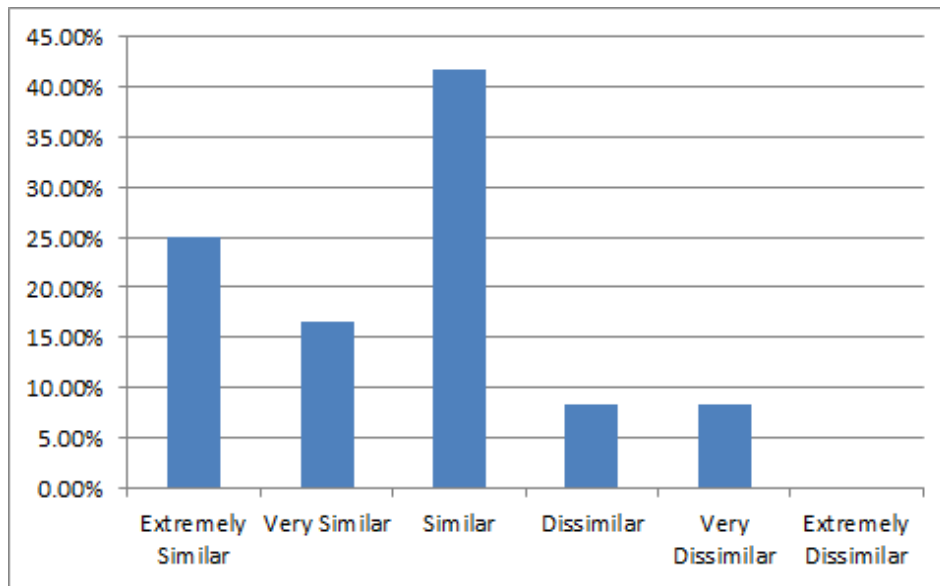
| Variable  | Values          |
|---|-----------------|
| $t$   | 2               |
| $\{u_1, u_2\}$ [if no sub-query] [WHERE and HAVING clauses] | $\{1, -1\}$     |
| $\{v_1, v_2, v_3\}$ [WHERE and HAVING clauses]              | $\{1, -1, 0\}$  |
| $\{u_1, u_2\}$ [if sub-query] [WHERE and HAVING clauses]    | $\{0.5, -0.5\}$ |
| $w_{rd\_i}$ [WHERE and HAVING clauses]                      | 1.5             |
| $\{k_1, k_2\}$ [WHERE and HAVING clauses]                   | $\{0, -0.5\}$   |
| $w_{wpr}$ [if no sub-query]                                 | 0.8             |
| $w_{wlp}$ [if no sub-query]                                 | 0.2             |
| $w_{hpr}$ [if no sub-query]                                 | 0.8             |
| $w_{hlp}$ [if no sub-query]                                 | 0.2             |
| $c$ [WHERE and HAVING clauses]                              | 3               |
| $x_{sp}$  | 0.7             |
| $W_{sl}$  | 10              |
| $W_{fr}$  | 10              |
| $W_{wh}$  | 35              |
| $W_{gb}$  | 14              |
| $W_{hw}$  | 25              |
| $W_{ob}$  | 10              |



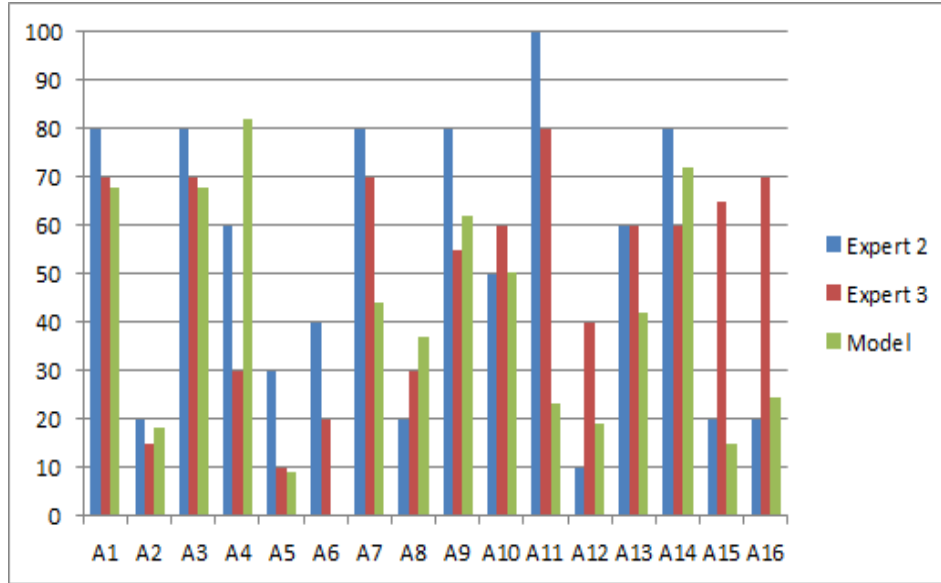
**Figure 4.1:** Comparison between scores by our model and 1st expert for data-set 1

**Table 4.5:** Table for similarity and dissimilarity counts of scores (in percentage) by 1st expert and our model for data-set 1

| Similarity           | Frequency (in percentage) |
|----------------------|---------------------------|
| Extremely Similar    | 25%                       |
| Very Similar         | 16.67%                    |
| Similar              | 41.67%                    |
| Dissimilar           | 8.33%                     |
| Very Dissimilar      | 8.33%                     |
| Extremely Dissimilar | 0%                        |



**Figure 4.2:** Chart for similarity and dissimilarity counts of score (in percentage) by 1st expert and our model for data-set 1



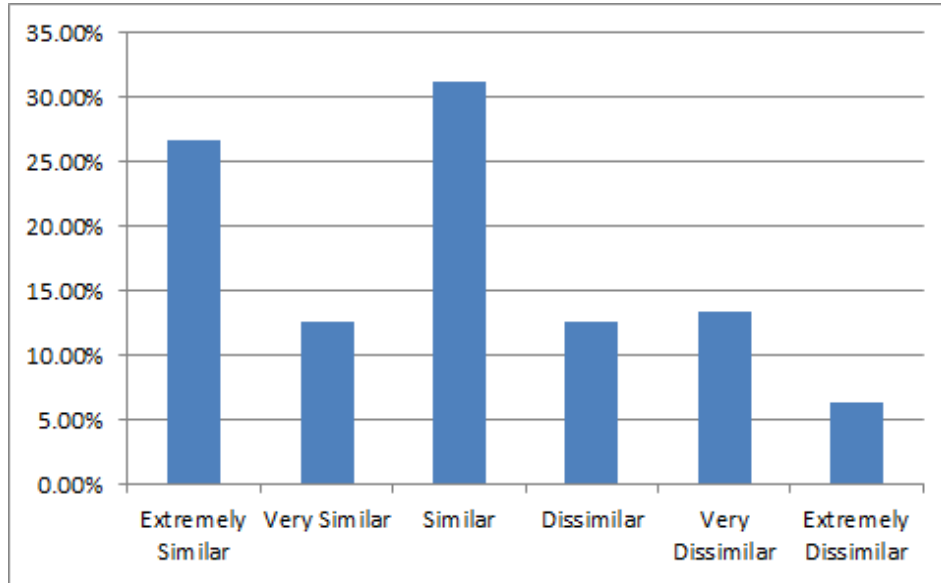
**Figure 4.3:** Comparison between scores by our model and experts for data-set 2

**Table 4.6:** Table for similarity and dissimilarity counts of scores (in percentage) by 2nd expert and our model for data-set 2

| Similarity           | Frequency (in percentage) |
|----------------------|---------------------------|
| Extremely Similar    | 25%                       |
| Very Similar         | 12.5%                     |
| Similar              | 31.25%                    |
| Dissimilar           | 12.5%                     |
| Very Dissimilar      | 12.5%                     |
| Extremely Dissimilar | 6.25%                     |

Now we present results for the second data set. We have collected evaluation scores from two experts (2nd expert and 3rd expert) and then compared the scores by the experts with scores computed using our model. We present the comparison between scores by the experts and our model in the form of clustered bar graph in Figure 4.3. We label the scores into the predefined categories showed in Table 4.3 and present them in percentage in Table 4.6, Table 4.7 and Figure 4.4 and Figure 4.5 respectively.

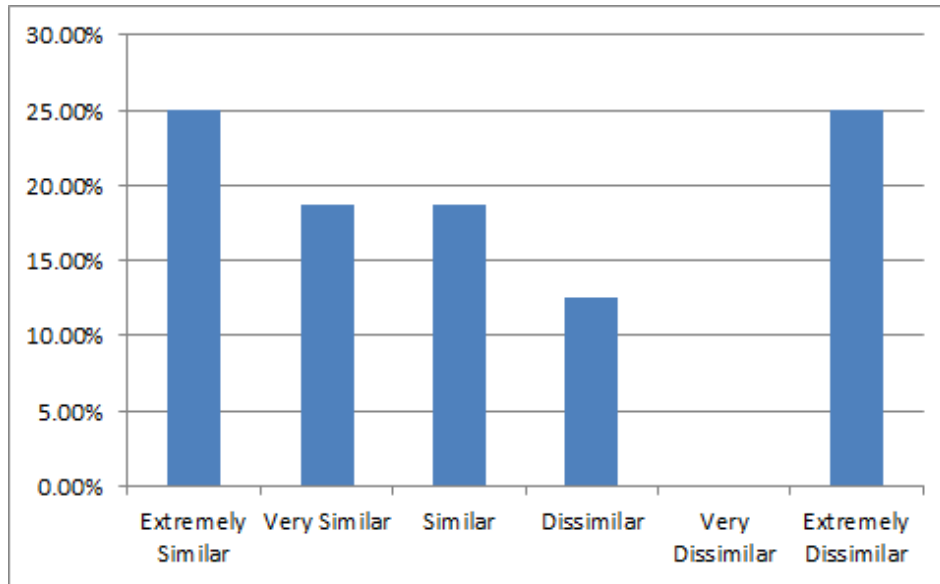
For data-set 2, our model generated scores are similar (Extremely Similar, Very Similar and Similar) to the scores by expert 2 for 68.75% of the cases.



**Figure 4.4:** Chart for similarity and dissimilarity counts of scores (in percentage) by 2nd expert and our model for data-set 2

**Table 4.7:** Table for similarity and dissimilarity counts of scores (in percentage) by 3rd expert and our model for data-set 2

| <b>Similarity</b>    | <b>Frequency (in percentage)</b> |
|----------------------|----------------------------------|
| Extremely Similar    | 25%                              |
| Very Similar         | 18.75%                           |
| Similar              | 18.75%                           |
| Dissimilar           | 12.5%                            |
| Very Dissimilar      | 0%                               |
| Extremely Dissimilar | 25%                              |



**Figure 4.5:** Chart for similarity and dissimilarity counts of scores (in percentage) by 3rd expert and our model for data-set 2

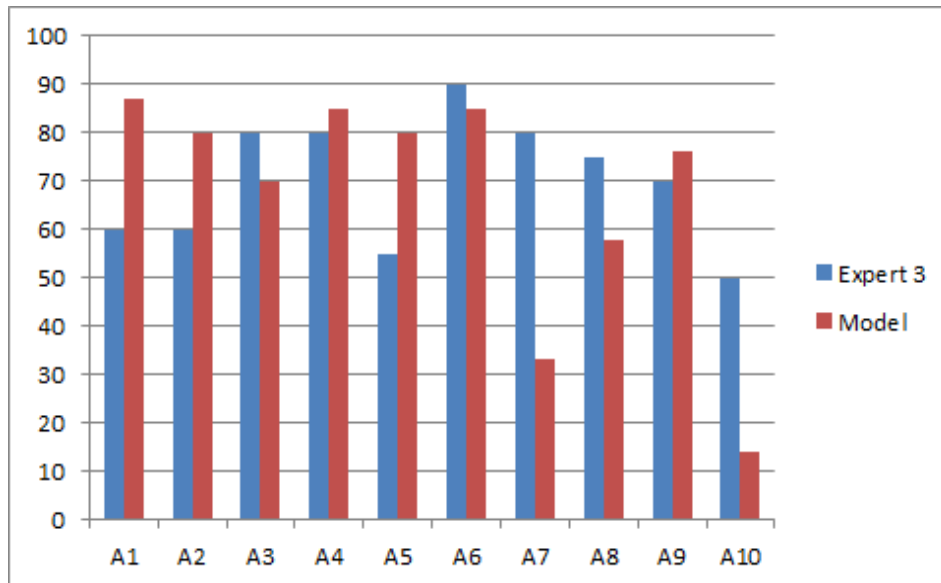
**Table 4.8:** Table for similarity and dissimilarity counts of scores (in percentage) by 3rd expert and our model for data-set 3

| Similarity           | Frequency (in percentage) |
|----------------------|---------------------------|
| Extremely Similar    | 20%                       |
| Very Similar         | 20%                       |
| Similar              | 20%                       |
| Dissimilar           | 20%                       |
| Very Dissimilar      | 0%                        |
| Extremely Dissimilar | 20%                       |

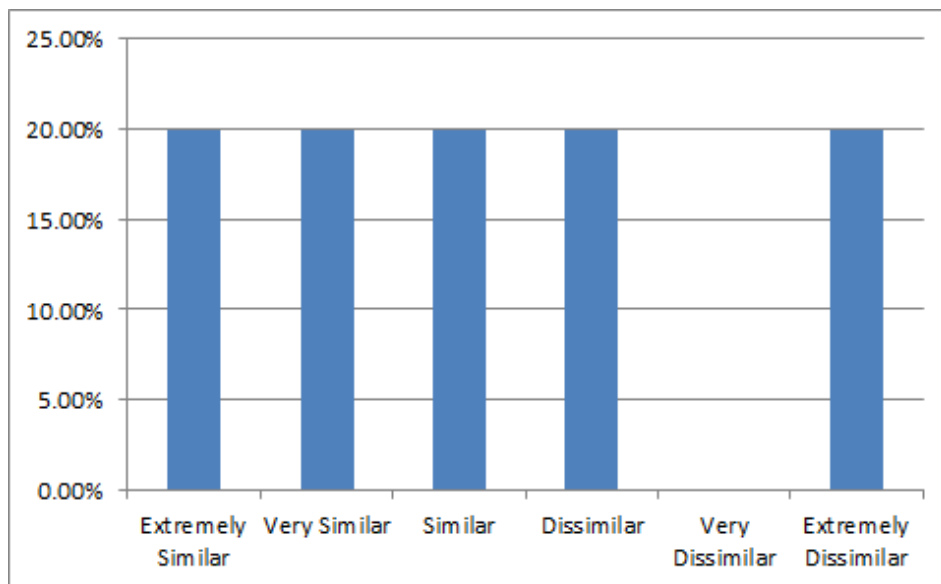
For data-set 2, our model based scores are similar (Extremely Similar, Very Similar and Similar) to the scores by 3rd expert for 62.5% of the cases.

We have collected scores from the 3rd expert for data-set 3. We present comparison between scores by 3rd expert and our model in Figure 4.6. We present the similarity and dissimilarity counts for data-set 3 in Figure 4.7 and Table 4.8.

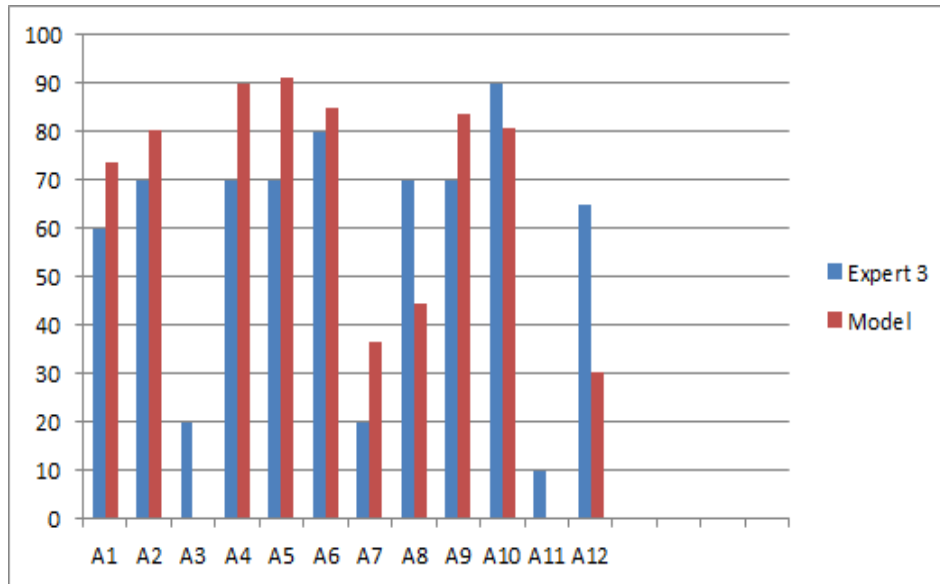
For data-set 3, our model based scores are similar (Extremely Similar, Very Similar and Similar) to the scores by 3rd expert for 60% of the cases.



**Figure 4.6:** Comparison between scores by our model and 3rd expert for data-set 3



**Figure 4.7:** Chart for similarity and dissimilarity counts of scores (in percentage) by 3rd expert and our model for data-set 3



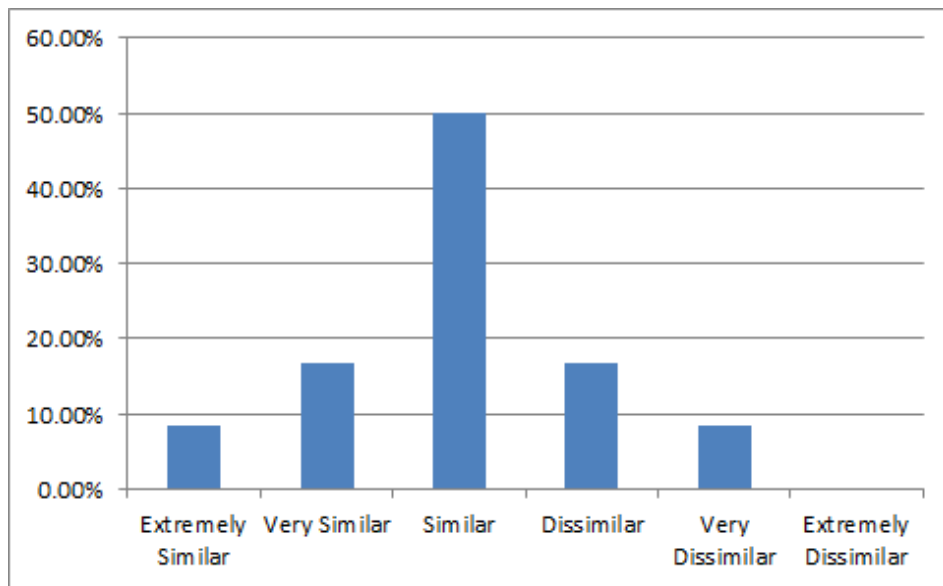
**Figure 4.8:** Comparison between scores by our model and 3rd expert for data-set 4

**Table 4.9:** Table for similarity and dissimilarity counts of scores (in percentage) by 3rd expert and our model for data-set 4

| <b>Similarity</b>    | <b>Frequency (in percentage)</b> |
|----------------------|----------------------------------|
| Extremely Similar    | 8.33%                            |
| Very Similar         | 16.67%                           |
| Similar              | 50%                              |
| Dissimilar           | 16.67%                           |
| Very Dissimilar      | 8.33%                            |
| Extremely Dissimilar | 0%                               |

We present comparison between scores by 3rd expert and our model for fourth data set in Figure 4.8. We present the similarity and dissimilarity counts for data-set 4 in Figure 4.9 and Table 4.9.

For data-set 4, our model based scores are similar (Extremely Similar, Very Similar and Similar) to the scores by 3rd expert for 75% of the cases.



**Figure 4.9:** Chart for similarity and dissimilarity counts of scores (in percentage) by 3rd expert and our model for data-set 4



# Chapter 5

## Conclusion

Despite the ongoing developments of various PBeL systems for teaching databases automatic partial evaluation is still not getting enough attentions. Most of the useful database e-learning/PBeL systems found in the literature adopted a common approach to automatic evaluation that evaluates answers as either correct or incorrect and ignore partial evaluation. One particular system that we found in the literature that employed partial evaluation had not described the particular model or framework used for evaluation.

In this report, we introduce a model for partial evaluation of SQL answers. The key idea behind our model is comparison of an SQL answer and the correct answer for the respective question and calculate a score that basically reflects how much similar the answered SQL expression is with respect to the reference SQL expression. We achieve this by calculating an individual score for every clause and then combining the individual scores along with appropriate weights for each clause to find the final score for the SQL answer.

We applied our model to 4 different datasets consisting of SQL expressions of various complexities including sub-queries. We collected scores for all the datasets from human experts and then we compared our model based scores with scores assigned by the experts. In the worst case, our model based scores were similar in 60% of the cases to the scores assigned by the expert while in best case our model based scores were similar to the expert-assigned scores for 83.34% of the cases. Based on the promising experimental results, we come to the conclusion that our model for partial evaluation is capable of assigning scores automatically to SQL answers of all complexities with high accuracies.

Weights play a crucial role in our partial evaluation model. The final score for an answered SQL expression is the weighted mean of the scores for all the individual clauses. In the model evaluation phase, we selected weights for different clauses such that the weights reflect the complexities of the respective clauses. We tuned the weights to achieve accuracy of partial scoring compared to the human graders. However we cannot state with certainty that our selected weights were optimal. This creates a scope for future study to use optimization techniques such as genetic algorithms to optimize the weights.

# Bibliography

- [1] W. Hung, D. H. Jonassen, and R. Liu, “Problem-based learning,” *Handbook of research on educational communications and technology*, vol. 3, pp. 485–506, 2008.
- [2] J. Kay, M. Barg, A. Fekete, T. Greening, O. Hollands, J. H. Kingston, and K. Crawford, “Problem-based learning for foundation computer science courses,” *Computer Science Education*, vol. 10, no. 2, pp. 109–128, 2000.
- [3] I. Richardson and Y. Delaney, “Problem based learning in the software engineering classroom,” in *2009 22nd Conference on Software Engineering Education and Training*, pp. 174–181, Feb 2009.
- [4] A. Hoque, M. M. Islam, M. I. Hossain, and M. F. Ahmed, “Problem-based e-learning and evaluation system for database design and programming in sql,” *International Journal of E-Education, E-Business, E-Management and E-Learning-IC4E*, pp. 537–542, 2013.
- [5] H. Kitaya and U. Inoue, “An online automated scoring system for java programming assignments,” *International Journal of Information and Education Technology*, vol. 6, no. 4, p. 275, 2016.
- [6] V. Karavirta and P. Ihanola, “Automatic assessment of javascript exercises,” in *CEUR Workshop Proceedings: WECU-2010 1st Educators’ Day on Web Engineering Curricula*, vol. 607, July 2010.
- [7] K. Ala-Mutka, T. Uimonen, and H.-M. Järvinen, “Supporting students in c++ programming courses with automatic program style assessment,” *Journal of Information Technology Education*, vol. 3, 2004.
- [8] S. Buyrukoglu, F. Batmaz, and R. Lock, “Semi-automatic assessment approach to programming code for novice students,” in *Proceedings of the 8th International Conference on Computer Supported Education*, pp. 289–297, 2016.
- [9] R. Singh, S. Gulwani, and A. Solar-Lezama, “Automated feedback generation for introductory programming assignments,” *ACM SIGPLAN Notices*, vol. 48, pp. 15–26, June 2013.

- [10] B. Chandra, M. Joseph, B. Radhakrishnan, S. Acharya, and S. Sudarshan, “Partial marking for automated grading of sql queries,” *Proceedings of the VLDB Endowment*, vol. 9, pp. 1541–1544, Sept. 2016.
- [11] J. Soler, F. Prados, I. Boada, and J. Poch, “A web-based tool for teaching and learning sql,” in *International Conference on Information Technology Based Higher Education and Training, ITHET*, 2006.
- [12] S. Sadiq, M. Orłowska, W. Sadiq, and J. Lin, “Sqlator: An online sql learning workbench,” *ACM SIGCSE Bulletin*, vol. 36, pp. 223–227, June 2004.
- [13] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques*. Elsevier, 2011.

# Appendices

**Table A.1: Data Set 1**

| Question  | Reference Query   | Answered Query   | Expert-1's Score (Out of 10) | Model's Score (Out of 10) |
|---|---|--|------------------------------|---------------------------|
| Find the names of all current/active Database Engineers.  | SELECT name<br>FROM EMPLOYEE<br>WHERE designation = 'Database Engineer' and active='yes'  | SELECT name<br>FROM EMPLOYEE<br>WHERE designation = 'Database Engineer'  | 5                            | 6.82                      |
| Find the names of all current/active Database Engineers.  | SELECT name<br>FROM EMPLOYEE<br>WHERE designation = 'Database Engineer' and active='yes'  | SELECT *<br>FROM EMPLOYEE<br>WHERE designation = 'Database Engineer'   | 4                            | 5                         |
| Find the number of current employees with salary more than 25000.   | SELECT COUNT(*)<br>FROM EMPLOYEE<br>WHERE salary > 25000 and active='yes'   | SELECT COUNT(*)<br>FROM EMPLOYEE<br>WHERE salary >= 25000 OR active='yes'  | 7                            | 8.1                       |
| Find the names and designations of employees who are currently working in the Application Development department.           | SELECT EMPLOYEE.name,EMPLOYEE.designation<br>FROM EMPLOYEE,DEPARTMENT,EMPLOYEE_DEPARTMENT<br>WHERE EMPLOYEE.id=EMPLOYEE_DEPARTMENT.e_id<br>AND DEPARTMENT.id=EMPLOYEE_DEPARTMENT.d_id<br>AND DEPARTMENT.name='Application Development'<br>AND EMPLOYEE_DEPARTMENT.active='Yes'  | SELECT EMPLOYEE.name, EMPLOYEE.designation<br>FROM EMPLOYEE,DEPARTMENT,EMPLOYEE_DEPARTMENT<br>WHERE EMPLOYEE.id=EMPLOYEE_DEPARTMENT.e_id<br>AND AND DEPARTMENT.name='Application Development'<br>AND EMPLOYEE_DEPARTMENT.active='Yes'                              | 5                            | 8.28                      |
| Find the names and designations of employees who are currently working in the Application Development department.           | SELECT EMPLOYEE.name, EMPLOYEE.designation<br>FROM EMPLOYEE,DEPARTMENT,EMPLOYEE_DEPARTMENT<br>WHERE EMPLOYEE.id=EMPLOYEE_DEPARTMENT.e_id<br>AND DEPARTMENT.id=EMPLOYEE_DEPARTMENT.d_id<br>AND DEPARTMENT.name='Application Development'<br>AND EMPLOYEE_DEPARTMENT.active='Yes' | SELECT EMPLOYEE.name, EMPLOYEE.designation<br>FROM EMPLOYEE,EMPLOYEE_DEPARTMENT<br>WHERE DEPARTMENT.name='Application Development'   | 2                            | 4.31                      |
| Find all the employees names, current or not, along with their respective departments.                                      | SELECT EMPLOYEE.name, DEPARTMENT.name<br>FROM EMPLOYEE,DEPARTMENT,EMPLOYEE_DEPARTMENT<br>WHERE EMPLOYEE.id=EMPLOYEE_DEPARTMENT.e_id<br>AND DEPARTMENT.id=EMPLOYEE_DEPARTMENT.d_id   | SELECT EMPLOYEE.name<br>FROM EMPLOYEE,EMPLOYEE_DEPARTMENT<br>WHERE EMPLOYEE.id=EMPLOYEE_DEPARTMENT.e_id  | 5                            | 5.31                      |
| Find all the employees names, current or not, along with their respective departments.                                      | SELECT EMPLOYEE.name, DEPARTMENT.name<br>FROM EMPLOYEE,DEPARTMENT,EMPLOYEE_DEPARTMENT<br>WHERE EMPLOYEE.id=EMPLOYEE_DEPARTMENT.e_id<br>AND DEPARTMENT.id=EMPLOYEE_DEPARTMENT.d_id   | SELECT EMPLOYEE.name, DEPARTMENT.name<br>FROM EMPLOYEE,DEPARTMENT,EMPLOYEE_DEPARTMENT  | 3                            | 3.64                      |
| Find the average salary of the employees currently working in the Software Quality Assurance department.                    | SELECT AVG(EMPLOYEE.salary)<br>FROM EMPLOYEE,DEPARTMENT,EMPLOYEE_DEPARTMENT<br>WHERE EMPLOYEE.id=EMPLOYEE_DEPARTMENT.e_id<br>AND DEPARTMENT.id=EMPLOYEE_DEPARTMENT.d_id<br>AND DEPARTMENT.name='Software Quality Assurance'<br>AND EMPLOYEE_DEPARTMENT.active='Yes'             | SELECT AVG(EMPLOYEE.salary)<br>FROM EMPLOYEE,DEPARTMENT,EMPLOYEE_DEPARTMENT<br>WHERE DEPARTMENT.name='Software Quality Assurance'  | 5                            | 4.91                      |
| Find the average salary of the employees working in the Software Quality Assurance department.                              | SELECT AVG(EMPLOYEE.salary)<br>FROM EMPLOYEE,DEPARTMENT,EMPLOYEE_DEPARTMENT<br>WHERE EMPLOYEE.id=EMPLOYEE_DEPARTMENT.e_id<br>AND DEPARTMENT.id=EMPLOYEE_DEPARTMENT.d_id<br>AND DEPARTMENT.name='Software Quality Assurance'<br>AND EMPLOYEE_DEPARTMENT.active='Yes'             | SELECT COUNT(EMPLOYEE.salary)<br>FROM EMPLOYEE,DEPARTMENT,EMPLOYEE_DEPARTMENT<br>WHERE EMPLOYEE.id=EMPLOYEE_DEPARTMENT.e_id<br>AND DEPARTMENT.name='Software Quality Assurance'<br>AND EMPLOYEE_DEPARTMENT.active='Yes'  | 8                            | 6.46                      |
| Find all employees with their designations that are manager of atleast one project.   | SELECT DISTINCT EMPLOYEE.name,EMPLOYEE.designation<br>FROM EMPLOYEE,PROJECT<br>WHERE EMPLOYEE.id = PROJECT.manager  | SELECT DISTINCT EMPLOYEE.name,EMPLOYEE.designation<br>FROM EMPLOYEE,PROJECT  | 2                            | 3.64                      |
| Find all employees with their designations that are manager of atleast one project.   | SELECT DISTINCT EMPLOYEE.name,EMPLOYEE.designation<br>FROM EMPLOYEE,PROJECT<br>WHERE EMPLOYEE.id = PROJECT.manager  | SELECT EMPLOYEE.name,EMPLOYEE.designation<br>FROM EMPLOYEE,PROJECT<br>WHERE EMPLOYEE.id = PROJECT.manager  | 9                            | 9.09                      |
| Find all employees with their designations that are currently assigned to the project 'HRM System development of ABC Corp'. | SELECT EMPLOYEE.name, EMPLOYEE.designation<br>FROM EMPLOYEE,PROJECT,EMPLOYEE_PROJECT<br>WHERE EMPLOYEE.id=EMPLOYEE_PROJECT.e_id<br>AND PROJECT.id=EMPLOYEE_PROJECT.p_id<br>AND PROJECT.title = 'HRM System development of ABC Corp'<br>AND EMPLOYEE_PROJECT.active = 'Yes'      | SELECT EMPLOYEE.name, EMPLOYEE.designation<br>FROM EMPLOYEE,PROJECT,EMPLOYEE_PROJECT<br>WHERE EMPLOYEE.id=EMPLOYEE_PROJECT.e_id<br>AND PROJECT.id=EMPLOYEE_PROJECT.p_id<br>AND PROJECT.title = 'HRM System development of ABC Corp'<br>AND EMPLOYEE.active = 'Yes' | 7                            | 8.73                      |

**Table A.2: Data Set 2**

| Question  | Reference Query   | Answered Query  | Expert -2's Score (Out of 10) | Expert-3's Score (Out of 10) | Model's Score (Out of 10) |
|---|---|---|-------------------------------|------------------------------|---------------------------|
| Find DId for DName = 'Mechanical Engineering'.  | select DId<br>from Lib_Department<br>where DName = 'Mechanical Engineering'   | select did<br>from lib_department<br>where dname= 'mechanical engineering'  | 8                             | 7                            | 6.8                       |
| Find DId for DName = 'Mechanical Engineering'.  | select DId<br>from Lib_Department<br>where DName = 'Mechanical Engineering'   | select dcodename<br>from lib_department<br>where Did=1  | 2                             | 1.5                          | 1.8                       |
| Find PAddress for PName 'SoftSolution'.   | select PAddress<br>from Lib_Publisher<br>where PName = 'SoftSolution'   | select paddress<br>from lib_publisher<br>where pname='softsolution'   | 8                             | 7                            | 6.8                       |
| Find PAddress for PName 'SoftSolution'.   | select PAddress<br>from Lib_Publisher<br>where PName = 'SoftSolution'   | select DateDiff<br>from lib_publisher<br>where PName= 'SoftSolution'  | 6                             | 3                            | 8.2                       |
| Find total number of publishers from USA.   | select count(*)<br>from Lib_Publisher<br>where PCountry = 'USA'   | select count(BookId)<br>from Lib_Publisher.lib_Book<br>where Lib_Publisher.PId=lib_Book.PId   | 3                             | 1                            | 0.9                       |
| Find total number of publishers from USA.   | select count(*)<br>from Lib_Publisher<br>where PCountry = 'USA'   | select sum(bookid)<br>from lib_book<br>where placeofpublication= 'USA'  | 4                             | 2                            | 0                         |
| Find the country and number of publishers in each country.  | select pcountry, count(*)<br>from Lib_Publisher<br>group by PCountry  | select count(*)<br>from lib_publisher<br>group by 'pcountry'  | 8                             | 7                            | 4.4                       |
| Find the country and number of publishers in each country.  | select pcountry, count(*)<br>from Lib_Publisher<br>group by PCountry  | select pcountry, pname<br>from lib_publisher  | 2                             | 3                            | 3.7                       |
| Find bookcopyid and book title of all the books having keyword "grammin".   | select bookcopyid,title<br>from lib_bookcopy,lib_book,lib_publisher<br>where lib_bookcopy.bookid=lib_book.bookid<br>and lib_book.pid=lib_publisher.pid<br>and bookkeywords like '%grammin%'   | select lbc.BookCopyId,lb.Title<br>from Lib_Book lb , Lib_BookCopy lbc<br>where lb.BookId=lbc.BookId<br>and lb.BookKeywords like 'grammin'   | 8                             | 5.5                          | 6.2                       |
| Find bookcopyid and book title of all the books having keyword "grammin".   | select bookcopyid,title<br>from lib_bookcopy,lib_book,lib_publisher<br>where lib_bookcopy.bookid=lib_book.bookid<br>and lib_book.pid=lib_publisher.pid<br>and bookkeywords like '%grammin%'   | select bookcopyid,title<br>from lib_bookcopy,lib_book<br>where title like '%grammin%'<br>and lib_book.bookid=lib_bookcopy.bookid  | 5                             | 6                            | 5.01                      |
| Find the title of the books whose purchase date is equal to that of book 'Database' in descending order of title.                               | select a.title<br>from lib_book a,lib_book b<br>where a.purchasedate=b.purchasedate<br>and b.title='Database'<br>order by title desc  | select Title<br>from Lib_Book<br>where purchaseDate in<br>(select purchaseDate<br>from Lib_Book<br>where Title = 'Database')  | 10                            | 8                            | 2.31                      |
| Find the title of the books whose purchase date is equal to that of book 'Database' in descending order of title.                               | select a.title<br>from lib_book a,lib_book b<br>where a.purchasedate=b.purchasedate<br>and b.title='Database'<br>order by title desc  | select purchasedate<br>from lib_book,lib_author<br>where lib_book.purchasedate='Database'<br>order by lib_book.title desc   | 1                             | 4                            | 1.9                       |
| Find title,subtitle and publisher name (pname) of the books that have been printed in years 1998 or 2000 in descending order of publisher name. | select title,subtitle,pname<br>from lib_bookcopy,lib_book,lib_publisher<br>where lib_bookcopy.bookid=lib_book.bookid<br>and lib_book.pid=lib_publisher.pid<br>and yearofprint in (1998,2000)<br>order by pname desc   | select Title,SubTitle,PName<br>from Lib_Book,Lib_Publisher<br>where Lib_Book.PId=Lib_Publisher.PId<br>and (YearOfPublication=1998<br>or YearOfPublication=2000)   | 6                             | 6                            | 4.2                       |
| Find title,subtitle and publisher name (pname) of the books that have been printed in years 1998 or 2000 in descending order of publisher name. | select title,subtitle,pname<br>from lib_bookcopy,lib_book,lib_publisher<br>where lib_bookcopy.bookid=lib_book.bookid<br>and lib_book.pid=lib_publisher.pid<br>and yearofprint in (1998,2000)<br>order by pname desc   | select Title,SubTitle,PName<br>from Lib_Book,Lib_Publisher,Lib_Bookcopy<br>where YearOfPrint=1998<br>or YearOfPrint=2000<br>and Lib_Publisher.PId=Lib_Book.PId<br>and Lib_Bookcopy.BookId=Lib_Book.BookId | 8                             | 6                            | 7.2                       |
| Find title and subtitle of the book those have the same pid as books of the publisher from pcity 'Chicago' in descending order of title.        | select title,subtitle<br>from lib_book<br>where lib_book.pid in<br>(select pid<br>from lib_publisher<br>where pcity='Chicago')<br>order by title desc   | select lib_book.title, lib_book.subtitle<br>from lib_book , lib_bookcopy,lib_publisher<br>where lib_book.bookid=lib_bookcopy.bookid<br>and lib_book.pid=lib_publisher.pid                                 | 2                             | 6.5                          | 1.5                       |
| Find Title and ISBN numbers of all books that belongs to 'CE' department but not to 'IPE' department.   | select Title, ISBN<br>from Lib_Book, Lib_BookDepartment, Lib_Department<br>where Lib_Book.bookid=Lib_BookDepartment.bookid<br>and Lib_BookDepartment.did=Lib_Department.did<br>and DcodeName='CE'<br>minus select Title, ISBN<br>from Lib_Book, Lib_BookDepartment, Lib_Department<br>where Lib_Book.bookid=Lib_BookDepartment.bookid<br>and Lib_BookDepartment.did=Lib_Department.did<br>and DcodeName='IPE' | select title,isbn<br>from lib_book 11,lib_department 12,lib_bookdepartment 13<br>where 11.bookid = 13.bookid<br>and 12.did = 13.did<br>and 12.dcodename = 'CE'<br>and 12.dcodename != 'IPE'               | 2                             | 7                            | 2.46                      |

**Table A.3: Data Set 3**

| Question  | Reference Query   | Answered Query   | Expert-3's Score (Out of 10) | Model's Score (Out of 10) |
|---|---|--|------------------------------|---------------------------|
| Find the total sales in the year 2017.  | SELECT sum(price)<br>FROM 'sales','customer','product'<br>WHERE 'customer'.id = 'sales'.customer_id'<br>AND 'product'.id = 'sales'.product_id'<br>AND 'sales'.year = 2017   | SELECT sum(price)<br>FROM 'sales','customer','product'<br>WHERE 'customer'.id = 'sales'.customer_id'<br>AND 'product'.id = 'sales'.product_id'<br>AND 'sales'.year = 2017<br>GROUP BY 'product'.brand'   | 6                            | 8.7                       |
| Find the total sales in the year 2017 for each brand.   | SELECT 'brand',sum(price) AS 'sales'<br>FROM 'sales','customer','product'<br>WHERE 'customer'.id = 'sales'.customer_id'<br>AND 'product'.id = 'sales'.product_id'<br>AND 'sales'.year = 2017<br>GROUP BY 'product'.brand'                     | SELECT 'brand',sum(price) AS 'sales'<br>FROM 'sales','customer','product'<br>WHERE 'customer'.id = 'sales'.customer_id'<br>AND 'product'.id = 'sales'.product_id'<br>AND 'sales'.year = 2017<br>AND 'sales'.year = 2017<br>GROUP BY 'product'.brand' | 6                            | 8                         |
| Show the total sales for the year 2017 for only those brands with minimum sales of 100000 Tk.                       | SELECT 'brand',sum(price)<br>FROM 'sales','customer','product'<br>WHERE 'customer'.id = 'sales'.customer_id'<br>AND 'product'.id = 'sales'.product_id'<br>AND 'sales'.year = 2017<br>GROUP BY 'product'.brand'<br>HAVING sum(price) >= 100000 | SELECT 'brand',sum(price)<br>FROM 'sales','customer','product'<br>WHERE 'customer'.id = 'sales'.customer_id'<br>AND 'product'.id = 'sales'.product_id'<br>AND 'sales'.year = 2017<br>AND sum(price) >= 100000<br>GROUP BY 'product'.brand'           | 8                            | 7                         |
| Show the total sales for the year 2017 for only those brands with minimum sales of 100000 Tk.                       | SELECT brand ,sum(price)<br>FROM 'sales','customer','product'<br>WHERE 'customer'.id = 'sales'.customer_id'<br>AND 'product'.id = 'sales'.product_id'<br>AND 'sales'.year = 2017<br>GROUP BY 'product'.brand'<br>HAVING sum(price) >= 100000  | SELECT brand ,sum(price)<br>FROM 'sales','customer','product'<br>WHERE 'customer'.id = 'sales'.customer_id'<br>AND 'product'.id = 'sales'.product_id'<br>AND 'sales'.year = 2017<br>HAVING sum(price) >= 100000                                      | 8                            | 8.5                       |
| Show the number of products sold for each brand.  | SELECT brand ,count(*)<br>FROM 'sales','customer','product'<br>WHERE 'customer'.id = 'sales'.customer_id'<br>AND 'product'.id = 'sales'.product_id'<br>AND 'sales'.year = 2017<br>GROUP BY 'product'.brand'                                   | SELECT brand ,count(*)<br>FROM 'sales','customer','product'<br>WHERE 'customer'.id = 'sales'.customer_id'<br>AND 'product'.id = 'sales'.product_id'<br>AND 'sales'.year = 2017   | 5.5                          | 8                         |
| Show the number of products the shop has for each brand.  | SELECT 'product'.brand',count(*)<br>FROM 'product'<br>GROUP BY 'product'.brand'   | SELECT 'product'.brand',sum(*)<br>FROM 'product'<br>GROUP BY 'product'.brand'  | 9                            | 8.5                       |
| Show the brand, model and price of the product that has the maximum price.  | SELECT 'brand','model',max(price)<br>FROM 'product'   | SELECT 'brand','model',price<br>FROM 'product'<br>HAVING 'price'>max(price)  | 8                            | 3.33                      |
| Show the brands along with the number of products that have atleast 3 products.                                     | SELECT 'brand', count(*)<br>FROM 'product'<br>GROUP BY 'brand'<br>HAVING count(*) >2  | SELECT 'brand', count(*) FROM 'product' GROUP BY 'brand' HAVING count(*) >3  | 7.5                          | 5.8                       |
| Show brands and the average prices of their products for only those brands whose average price do not exceed 30000. | SELECT 'brand',avg(price)<br>FROM 'product'<br>GROUP BY 'brand'<br>HAVING avg(price) <=30000  | SELECT 'brand',avg(price)<br>FROM 'product'<br>HAVING avg(price) <=30000   | 7                            | 7.63                      |
| Show brands and the average prices of their products for only those brands whose average price do not exceed 30000. | SELECT 'brand',avg(price)<br>FROM 'product'<br>GROUP BY 'brand'<br>HAVING avg(price) <=30000  | SELECT 'brand',avg(price)<br>FROM 'product'<br>WHERE avg(price) <=30000  | 5                            | 1.4                       |

**Table A.4: Data Set 4**

| Question   | Reference Query  | Answered Query   | Expert-3's Score (Out of 10) | Model's Score (Out of 10) |
|--|--|--|------------------------------|---------------------------|
| Find publisher id, purchase date and total price in taka (pricetaka) of books purchased in different dates for those publishers who lives in the same country as the publishers who have yahoo email address. Result must be in descending order of price. | select lib.publisher.pid,purchasedate,sum(pricetaka)<br>from lib_bookcopy.lib_book.lib_publisher<br>where lib_bookcopy.bookid=lib_book.bookid<br>and lib_book.pid=lib_publisher.pid<br>and pcountry in<br>(select pcountry<br>from lib_publisher<br>where pemail like '%yahoo%')<br>group by lib_publisher.pid, purchasedate<br>order by sum(pricetaka) desc | select p.pid,b.purchasedate,c.pricetaka<br>from lib_book b,lib_publisher p,lib_bookcopy c<br>where c.bookid=b.bookid<br>and b.pid=p.pid and p.pcountry in<br>(select pcountry<br>from lib_publisher<br>where pemail like '%@yahoo.com')<br>order by pricetaka desc | 6                            | 7.38                      |
| <b>Continued on next page</b>  |  |  |                              |                           |

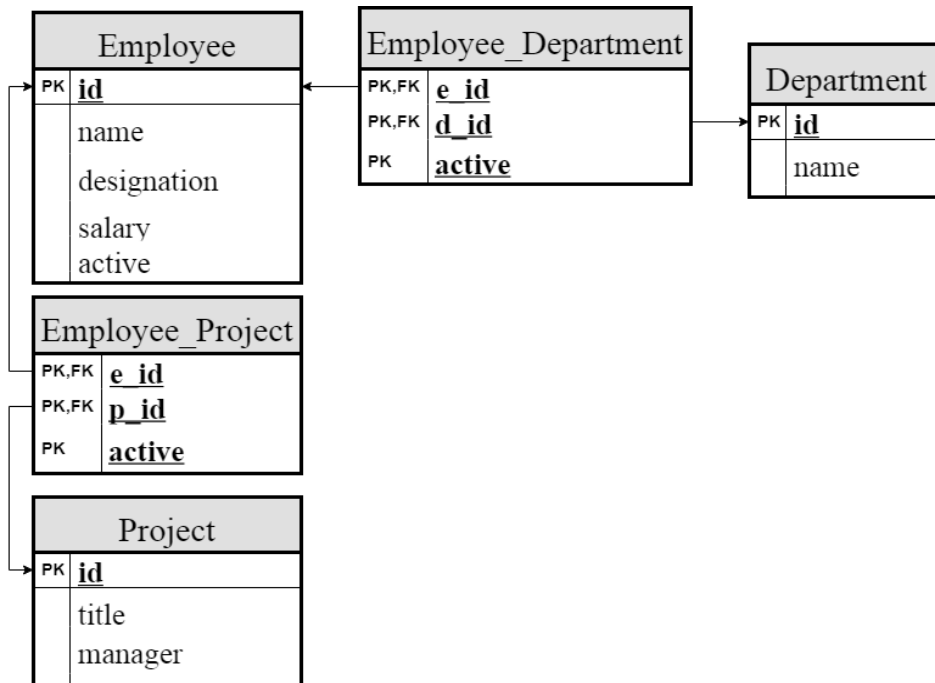
**Table A.4: Data Set 4**

| Question   | Reference Query   | Answered Query   | Expert-3's Score (Out of 10) | Model's Score (Out of 10) |
|--|---|--|------------------------------|---------------------------|
| Find Title of all books of 'EEE' department that were published in the same year or after the year of publication of book titled 'Database'.   | select Title<br>from Lib_Book, Lib_BookDepartment, Lib_Department<br>where Lib_Book.bookid=Lib_BookDepartment.bookid<br>and Lib_BookDepartment.did=Lib_Department.did<br>and DCodeName='EEE'<br>and yearofpublication >=<br>(select yearofpublication<br>from Lib_Book<br>where title='Database')   | SELECT TITLE<br>FROM Lib_Book, Lib_BookDepartment, Lib_Department<br>WHERE Lib_BookDepartment.BookId = Lib_Book.BookId<br>AND Lib_BookDepartment.Did = Lib_Department.Did<br>AND Lib_Department.DCodeName = 'EEE'<br>AND YearOfPublication ><br>(SELECT distinct YearOfPublication<br>FROM Lib_Book, Lib_BookDepartment, Lib_Department<br>WHERE Lib_BookDepartment.BookId = Lib_Book.BookId<br>AND Lib_BookDepartment.Did = Lib_Department.Did<br>AND Title = 'Database') | 7                            | 8.03                      |
| Find Title of all books of 'EEE' department that were published in the same year or after the year of publication of book titled 'Database'.   | select Title<br>from Lib_Book, Lib_BookDepartment, Lib_Department<br>where Lib_Book.bookid=Lib_BookDepartment.bookid<br>and Lib_BookDepartment.did=Lib_Department.did<br>and DCodeName='EEE'<br>and yearofpublication >=<br>(select yearofpublication<br>from Lib_Book<br>where title='Database')   | select DName,count(BookId)<br>from Lib_BookDepartment,Lib_Department<br>where Lib_Department.DID=Lib_BookDepartment.DID<br>group by Lib_Department.Dname<br>having count(BookId)>=3  | 2                            | 0.02                      |
| Find Title of all books with pricebase greater than average pricebase of 'EEE' department.   | select Title<br>from Lib_Book<br>where pricebase ><br>(select avg(pricebase)<br>from Lib_Book, Lib_BookDepartment, Lib_Department<br>where Lib_Book.bookid=Lib_BookDepartment.bookid<br>and Lib_BookDepartment.did=Lib_Department.did<br>and DcodeName='EEE')   | select Lib_Book.Title<br>from Lib_Book<br>where Lib_Book.PriceBase ><br>(select avg(PriceBase)<br>from Lib_Book,Lib_BookDepartment,Lib_Department<br>where Lib_Book.BookId = Lib_BookDepartment.BookId<br>and Lib_BookDepartment.Did = Lib_Department.DID<br>and Lib_Department.DName='EEE')   | 7                            | 8.98                      |
| Find the name of those borrowers and book titles booked by them who has booked some book before the date of "Mr. Kamal" has booked some book. in descending order of both name of the borrowers and title. | select bName, title<br>from lib_book, Lib_Booking, Lib_Borrower<br>where Lib_book.bookId = Lib_Booking.bookID<br>and Lib_Booking.bid = Lib_Borrower.bid<br>and BookingDate <<br>(select min(BookingDate)<br>from Lib_Booking, Lib_Borrower<br>where Lib_Booking.bid = Lib_Borrower.bid<br>and bName = 'Mr. Kamal')<br>order by bname desc, title desc | select BNAME , TITLE<br>from LIB_BORROWER a,LIB_BOOKING b,LIB_BOOK c<br>where a.BID=b.BID<br>and b.BOOKID=c.BOOKID<br>and BOOKINGDATE ><br>(select BOOKINGDATE<br>from LIB_BORROWER a,LIB_BOOKING b<br>where b.BID=a.BID<br>and BNAME='Mr.Kamal')  | 7                            | 9.11                      |
| Find the number of book which was published in the same year of the book "Database".   | select count(bookID)<br>from lib_book<br>where yearofpublication =<br>(select yearofpublication<br>from lib_book<br>where title = 'Database')   | select count(bookid)<br>from lib_book<br>where yearofpublication =<br>(select yearofpublication<br>from lib_book<br>where title = 'database')  | 8                            | 8.47                      |
| Find the title and pricebase of books which were not published in 'USA' and which have a pricebase greater than twice the average pricebase of all books published in 'USA'.                               | select title,pricebase<br>from lib_book<br>where placeofpublication !='USA'<br>and pricebase ><br>(select 2*avg(pricebase)<br>from lib_book<br>group by placeofpublication<br>having placeofpublication='USA')  | select Title, PriceBase<br>from Lib_Book<br>where PlaceOfPublication='Singapore'<br>or PlaceOfPublication='India'<br>and PriceBase > 800   | 2                            | 3.64                      |
| Find the title and pricebase of books which were not published in 'USA' and which have a pricebase greater than twice the average pricebase of all books published in 'USA'.                               | select title,pricebase<br>from lib_book<br>where placeofpublication !='USA'<br>and pricebase ><br>(select 2*avg(pricebase)<br>from lib_book<br>group by placeofpublication<br>having placeofpublication='USA')  | select title,pricebase<br>from lib_book,lib_publisher<br>where lib_book.pid=lib_publisher.pid<br>and pcountry <>'USA'<br>and pricebase >(select 2*avg(pricebase)<br>from lib_book,lib_publisher<br>where lib_book.pid=lib_publisher.pid<br>and pcountry = 'USA')   | 7                            | 4.45                      |
| Find distinct isbn,title,publisher name of the most recently printed books in the library.<br>Output : isbn,title,pname<br>Order by : isbn,title,pname   | select distinct<br>isbn,title,pname<br>from lib_bookcopy,lib_book,lib_publisher<br>where lib_bookcopy.bookid=lib_book.bookid<br>and lib_book.pid=lib_publisher.pid<br>and yearofprint in<br>(select max(yearofprint)<br>from lib_bookcopy)<br>order by isbn,title ,pname  | select distinct<br>isbn, title, pname<br>from lib_bookcopy, lib_book, lib_publisher<br>where lib_bookcopy.bookid= lib_book.bookid<br>and lib_book.pid= lib_publisher.pid<br>and yearofprint=<br>(select min(yearofprint)<br>from lib_bookcopy)<br>order by isbn, title, pname  | 7                            | 8.35                      |
| Find distinct isbn,title,publisher name of the most recently printed books in the library.<br>Output : isbn,title,pname<br>Order by : isbn,title,pname   | select distinct<br>isbn,title,pname<br>from lib_bookcopy,lib_book,lib_publisher<br>where lib_bookcopy.bookid=lib_book.bookid<br>and lib_book.pid=lib_publisher.pid<br>and yearofprint in<br>(select max(yearofprint)<br>from lib_bookcopy)<br>order by isbn,title ,pname  | select distinct<br>isbn,title,PName<br>from lib_book,Lib_Publisher,Lib_BookCopy<br>where lib_book.bookid=Lib_BookCopy.bookid<br>and Yearofprint=<br>(select max(Yearofprint)<br>from Lib_BookCopy)<br>order by isbn,title,PName  | 9                            | 8.09                      |
| Continued on next page   |   |  |                              |                           |



**Table A.4: Data Set 4**

| Question   | Reference Query  | Answered Query  | Expert-3's Score (Out of 10) | Model's Score (Out of 10) |
|--|--|---|------------------------------|---------------------------|
| Find DCodeName of departments having books with total pricebase <sub>i</sub> =1500 and the departments do not have a book titled 'Database'. | <pre>select DCodeName from Lib_Book, Lib_BookDepartment, Lib_Department where Lib_Book.bookid=Lib_BookDepartment.bookid and Lib_BookDepartment.did=Lib_Department.did group by DcodeName having sum(pricebase)&gt;1500 minus select DCodeName from Lib_Book, Lib_BookDepartment, Lib_Department where Lib_Book.bookid=Lib_BookDepartment.bookid and Lib_BookDepartment.did=Lib_Department.did and Title='Database'</pre> | <pre>select * from DCodeName</pre>  | 1                            | 0                         |
| Find DCodeName of departments having books with total pricebase <sub>i</sub> =1500 and the departments do not have a book titled 'Database'. | <pre>select DCodeName from Lib_Book, Lib_BookDepartment, Lib_Department where Lib_Book.bookid=Lib_BookDepartment.bookid and Lib_BookDepartment.did=Lib_Department.did group by DcodeName having sum(pricebase)&gt;1500 minus select DCodeName from Lib_Book, Lib_BookDepartment, Lib_Department where Lib_Book.bookid=Lib_BookDepartment.bookid and Lib_BookDepartment.did=Lib_Department.did and Title='Database'</pre> | <pre>select DCodeName from Lib_Book.Lib_BookDepartment,Lib_Department where Lib_BookDepartment.Did = Lib_Department.Did and Lib_Book.BookId=Lib_BookDepartment.BookId and Title not in ('Database') group by DCodeName having sum(pricebase)&gt;=1500</pre> | 6.5                          | 3.01                      |



**Figure A.1: Database Schema for Data Set 1**

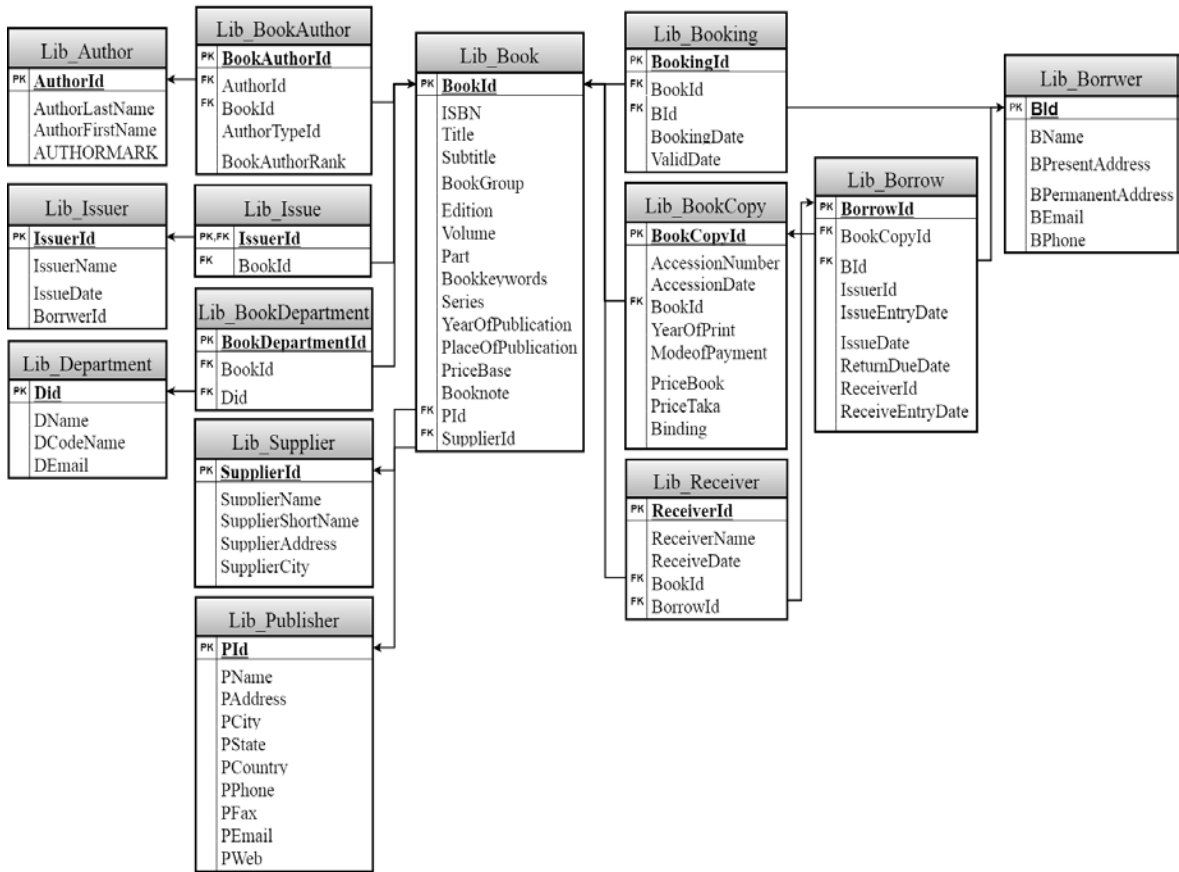


Figure A.2: Database Schema for Data Set 2 and Data Set 4

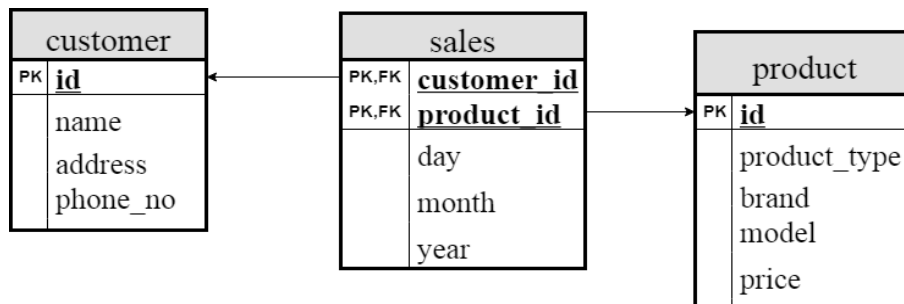


Figure A.3: Database Schema for Data Set 3