Equivalence of Problems in Problem Based e-Learning of Database

Submitted by
Md. Rasel Uddin
Student ID: 100705003P

A thesis submitted to the Department of Computer Science and Engineering in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE IN ENGINEERING IN
COMPUTER SCIENCE AND ENGINEERING

Supervised by
Dr. A. S. M. Latiful Hoque
Professor, Department of CSE, BUET

Department of Computer Science and Engineering
BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY
Dhaka, Bangladesh
February, 2013

The thesis "**Equivalence of Problems in Problem Based e-Learning of Database",** submitted by Md. Rasel Uddin, Roll No. 100705003P, Session: October 2007, to the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, has been accepted as satisfactory for the partial fulfilment of the requirements for the degree of Master of Science in Engineering (Computer Science and Engineering) and approved as to its style and contents. Examination held on March 16, 2013.

Board of Examiners

Declaration

I, hereby, declare that the work presented in this thesis is the outcome of the investigation performed by me under the supervision of Dr. A. S. M. Latiful Hoque, Professor, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka. I also declare that no part of this thesis and thereof has been or is being submitted elsewhere for the award of any degree.

(Md. Rasel Uddin)

Acknowledgement

First I express my heartiest thanks and gratefulness to Almighty Allah for His divine blessings, which made me possible to complete this thesis successfully.

I feel grateful to and wish to acknowledge my profound indebtedness to Professor Dr. A. S. M. Latiful Hoque, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology. Deep knowledge and keen interest of Professor Dr. A. S. M. Latiful Hoque in the field of e-Learning influenced me to carry out this thesis. His endless patience, scholarly guidance, continual encouragement, constructive criticism and constant supervision have made it possible to complete this thesis.

I also express my gratitude to Professor Dr. A. S. M. Latiful Hoque, Head of the Department of Computer Science and Engineering, BUET for providing me enough lab facilities to make necessary experiments of my research in the Graduate lab of BUET.

I would like to thank the members of the Examination committee, Dr. Mohammad Eunus Ali, Associate Professor, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dr. Tanzima Hashem, Assistant Professor, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology and Dr. M Abdul Awal, Professor, Department of Electrical Engineering & Computer Science, North South University, Dhaka for their helpful suggestions and careful review of this thesis.

I would like to convey gratitude to all my course teachers whose teaching helps me a lot to start and complete this thesis work.

Lastly I am also grateful to my family and colleagues for giving me continuous support.

Abstract

Problem-based learning (PBL) in engineering education is an important research area. Several works have been done to express PBL methodologies, problem design, implementation of learning environment, support PBL learning, evaluate performance, group assignment and others. Problem solving is the main goal of education. Problem-based learning is a way to learn what is needed to solve a problem, how can a solution be obtained quickly, precisely and professionally. To achieve the goal of problem-based learning, problem design and assign same level of problems among the students are important in engineering classroom environment.

SQL is a major part in Database course. In problem-based e-Learning of SQL, it is essential to find out the equivalence of a SQL problems to assign the set of problems to a set of students. This is necessary for equal judgment of the performance of individual students. We have developed a complexity model to find out the equivalence of problems for problem based e-learning of database. In this model, complexity of problems is found by parsing the given solution of the problem in top down approach.

We have applied our model to well known SQL Learning and Evaluation System (SQL-LES). We have compared our calculated complexity value with the complexity value in the question bank of SQL-LES assigned by the SQL experts and found that in most case our model generate similar complexity value as SQL-LES. Application of our model will reduce the instructor workload in SQL-LES.

Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Background

Problem based learning (PBL) is an instructional method in which students learn through facilitated problem solving. PBL is important and essential in engineering education. Technology has been changing over time. To fetch the changing technological problem, engineering students need to be technically competent, self learner, communicative and think creatively. Traditional learning does not fulfil these requirements. Traditional learning environment based on memorized knowledge. Students can't apply their knowledge what they have learned from traditional method in real life problem.

In PBL, teacher acts to facilitate the learning process rather than to provide knowledge. The goals of PBL include helping students develop flexible knowledge, effective problem solving skills, self-directed learning (SDL) skills, effective collaboration skills, and intrinsic motivation. Facilitator assigns problem to student for learning. Students work in collaborative groups to identify what they need to learn in order to solve the problem. They engage in self-directed learning and then apply their new knowledge to the problem and reflect on what they have learned and the effectiveness of the strategies employed.

In PBL, teacher fetches some problems to assign similar level of problems among the students. It is very difficult to find out the complexity value of problems. Using the complexity value, teacher can determine equivalence of problems. The equivalence of problems means that the complexity to solve the problem is within a specified boundary. Those problems fall into a specified boundary are all equivalent problems and any of the problem can be assigned to any student and seem to have equal judgement. So it is necessary to develop a complexity model to find out complexity value that can be used to identify problem equivalence.

1.2 Problem Definition

Most important criteria for Problem-based learning is to define the problems and distribute them among the students. Existing problem-based learning focus on the PBL methodology, learning content, learning environment, problem design, e-learning for PBL, tools to submit report, discussion forum, problem design, monitor student activity and others. Existing systems do not focus on the complexity of the problem and distribute among the students. Simply PBL facilitators choose problems and assign them to the student for learning. Where some students complain that the problem is very simple for learning and other students complain that the problem is very complex to solve within limited time.

Database is the core course in Computer Science and Engineering. SQL is an important part of Database. In Problem-based Learning and Evaluation of SQL, students are assigned multiple assignments with a varying complexity. Existing SQL Learning and Evaluation systems assign the complexity values of SQL problems manually based on domain knowledge of the instructors. If the class size is large multiple instructors produce multiple assignments then it is difficult to have an equivalence of assignments. Students' performance sometime varies because of the dissimilarities of the assignments given by different instructors. At the same time, if the SQL question bank contains hundreds of questions, it is extremely difficult to obtain a global complexity value of each SQL problem to reuse the problems.

1.3 Objectives

The objectives of the thesis are to:

- design a complexity model to find out complexity value of a SQL problem,
- apply the proposed complexity model on Problem-based e-Learning of Database,
- evaluate the performance of the model by applying the model in an SQL-Learning and Evaluation System.

1.4 Overview of the Thesis

In problem based learning, teacher assigns problem among the students at the beginning of PBL session. Similar level of problem defines and distributions among the students are important in problem based learning. Problem equivalence can be found out by analyzing the problem in top-down fashion. We have proposed Complexity Model to find out complexity value of SQL problems. By comparing complexity value, we have found out the equivalence problems.

Complexity Model parses the problem in top-down fashion to find out the required domain knowledge. Through top-down analysis, PBL problem has been divided into sub-problems.

Then each sub-problem analyzes to find out critical, meaningful and complex item to solve the problem. All discovered item then mark with some weighted value. To find out problem complexity value, weighted value has been calculated using different mathematical formula.

In Problem based Learning and Evaluation of SQL, students are assigned multiple assignments with varying complexities. We have discussed details about SQL operations like CREATE, INSERT, DELETE, UPDATE and SELECT. SQL complexity depends on how much domain and general knowledge required to solve a problem. We have analyzed those SQL operations to find out the used domain knowledge. Same item can use in different place in SQL statement. Based on the used position and item type, we have assigned some complexity value to that item. Similar items can repeat within a problem several times which does not seem to increase the complexity of a problem. To curb complex value, we have used logarithm function. Finally we have applied our newly developed system on existing SQL-LES question banks as a case study and found that in most case our model generate similar complexity value as SQL-LES.

Existing works does not focus on the equivalence of problems. We have proposed Complexity Model to find out equivalence of problems using the complexity value of different problems.

1.5 Organization of the Thesis

The thesis is organized as follows:

In Chapter 2, we have discussed details about existing works on Problem based Learning in engineering education.

In Chapter 3, we have discussed details about the complexity model and analysis of the model, problem equivalence and mathematical formula to find out problem complexity value. Database has many SQL operations. We have analyzed those operations in details to find out problem complexity value.

In Chapter 4, we have evaluated the Complexity Model using real SQL problems for different operations. We have calculated complexity value of different SQL operations to find out the equivalence of problems. To calculate complexity value of a SQL problem, we have defined complexity value of individual keyword, function, predicate and others. Finally we have applied our model on existing SQL-LES question bank and shown the result.

In the Chapter 5, we have concluded this thesis with contributions and further research directions.

Chapter 2
Literature Review

Problem based Learning (PBL) is a blended learning environment, a combination of self-directed learning and collaborative learning [1]. Learners need to know methods, techniques and standard practices which help to develop skill, positive learning aptitudes and get valuable experience. Problem based e-Learning (PBeL) can support and complement the problem-based learning model with knowledge transfer [2], [3]. The design of problems, finding the complexity of the problems to assign students individually or in a group and evaluation of the problems are the challenging issues in PBL Systems.

Many researchers have focused on teaching strategy, student learning, tutor roles, student roles, grading, and group distribution [1], [2], [3]. Meaningful learning and problem solving can only be acted out in a certain learning environment [4]. Hung [5] proposed the 3C3R paper based model, a systematic conceptual framework for guiding the design of effective and reliable PBL problems. PBL activities like report submission, group discussion, construction learning content, student feedback and assessment has supported with web-based learning environment [6], [7]. A number of research and development have been done on e-Learning to support problem based learning [8], [9], [10].

Hoque [11] has developed SQL-LES to teach SQL query by assigning SQL problem to students from question bank. Teachers choose problem based on the complexity value of that problem. Teachers assign the complexity value during the creation of question bank by analyzing of the SQL queries and the result.

2.1 What and How do Students Learn in PBL

Hmelo-Silver described what and how student learn in Problem-based learning (PBL) [1]. PBL is part of this tradition of meaningful, experiential learning. In PBL, students learn by solving problems and reflecting on their experiences. Students learn through the experience of solving problems, they can learn both content and thinking strategies. PBL is an instructional method in which students learn through facilitated problem solving. Students work in collaborative groups to identify what they need to learn in order to solve a problem. They engage in self-directed learning (SDL) and then apply their new knowledge to the problem and reflect on what they learned and the effectiveness of the strategies employed. This approach focused, experiential learning organized around the investigation, explanation, and resolution of meaningful problems. The PBL learning cycle is enacted through the tutorial process that begins with the presentation of a problem and ends with student reflection. A PBL tutorial session begins by presenting a group of students with minimal information about a complex problem.

2.2 PBL for Software Engineering Course

Ming et al. applied PBL approach in a course “Advance Software Engineering” in engineering education [2]. This approach is conducted a blended learning environment, a combination of a face-to-face learning environment and e-Learning environment. A set of integrated projects were selected as stimulus to learning. Both inter- and intra-group collaborative learning are encouraged. A survey conducted in the end of the course showed that students accept the problem-based learning quiet well, and their academic achievements were also better than expected.

Ming has divided students into group of different size. Each group is guided by teacher assistant. At the beginning of class, project concepts, submission deadline and other rules are introduced. The students were asked to follow the guideline and develop the project in an iterative and incremental way. A credit system was introduced to differentiate the grades of students. All credits are given to teams - not individuals. At the end of the course, each team proposed a credit distribution for its members, based on their contributions to the project. It is challenging to develop a course based on PBL. One of the challenges encountered was projects distribution. While some students complained that the project is too simple, some others were unconfident that they can get through the project.

2.3 PBL in Software Engineering Classroom

Non-traditional teaching method was introduced for inexperienced students to understand in software engineering classroom by Ita et al. [3]. It present factors which should exists in pure problem-based learning. This problem-based learning class was observed and analyzed by the second author Yvonne. The analysis presented focuses on the problem-based learning factors, how they were implemented in class, and the strengths and weaknesses of the use of problem-based learning in this way. The authors also discuss how the teaching could be improved through modifying the teaching method for a future class in which problem-based learning will be used. Yvonne attended class during different session to find out how the students were handling the problem in class and how the lecturer was facilitating. Yvonne compared the problem, the facilitation and the participation of the class with standard PBL methodology. This is expected to improve the understanding of the PBL methodology, the role of the student in determining their learning issues, the facilitation process, the importance of the student's role in the team, the assessment methodology and problem development. Fundamental beliefs will be challenged. Building a comprehensive PBL community requires determination and commitment from all levels – student, faculty and management – to make it work.

2.4 PBL for Engineering Education

The development of positive learning aptitudes on engineering students has been carried out with the help of the problem based learning (PBL) methodology by Lacuesta et al. [4]. Solutions of medium-high complexity problems by students make them work on the development of different skills. The teaching model turns into a significant and autonomous learning model where students are conscious of their compromise with this process (learning). Competences can be regarded as skills or abilities to understand and use knowledge, solve problems, use tools or technologies, learn in an autonomous way, research and think with initiative and creativity, communicate, cooperate, and so on.

To implement PBL methodology, the development of the project/problem presented by the lecturer will have to allow students to integrate the contents of all of them. The proposed project must have the characteristics of an appropriate problem of PBL. The use of a virtual learning platform will be also recommended to improve both the lecturer-student and lecturer-team interaction using different tools as chat, email, forums, private intranets, etc. It is very suitable and advisable that documentation produced by students is homogeneous, maintaining the same format. The development of a PBL experience lies on the following activities to be carried out by lecturers and/or students:

- Explanation of the experience to develop.
- Initial explanation of theoretical concepts.
- Allocation of groups, projects and roles.
- Continuous monitoring and evaluation.
- Final Assessment.

As regards abilities and skills developed by the students, the results obtained coming from the last experience showed that the abilities and skills more developed were: self learning, knowledge integration, oral and written communication, critical capacity, team work and initiative. Similar project/problem distribution and assessment is very important and challenging task in PBL methodology.

2.5 Framework for PBL Environments

Qian *et al*. proposed a framework for designing problem-based learning environment (PBLE) [5]. It consist three layers--a goal layer, a base layer and a core layer. Activity (problem/task and its context) with various interpretative and intellectual support systems surrounding it is the authentic elements of the environment. Related elements include information resources, tools and scaffoldings can support understanding of the problem and suggest possible solutions and help learners to interpret and manipulate aspects of the problems. The relation of teachers, learner, as well as the virtual learning environment (VLE) is an important design issue for deigning problem-based learning environment. PBLE concentrate on community building. Communities enable the learner to negotiate and co-construct meaning for the problem; and also to help teachers to implement the PBLEs.

2.6 3C3R

Hung developed conceptual framework for designing problem in problem-based learning [6]. Well-designed problems are crucial for the success of problem-based learning (PBL). The 3C3R model comprises two classes of components: core components and processing components. Core components—including content, context, and connection—support content and conceptual learning, while processing components—consisting of researching, reasoning, and reflecting—concern students' cognitive processes and problem-solving skills. To optimize and maximize the effects of PBL, the quality of the problems is vital. Research is needed to evaluate and validate the 3C3R model in terms of its comprehensiveness and conceptual soundness in guiding instructional designers and educators to design effective PBL problems. Further studies are needed to examine whether the 3C3R model can sufficiently address these different requirements for solving different types of problems as well as the interaction between types of problems and the components of the 3C3R model.

2.7 e-Learning System for Problem-based Education

Lian *et al.* developed e-Learning system for problem-based education [10]. It provides the learning environment, tools, resources and management for e-learning such as support to the creation of scenarios, information integration, resource sharing, and collaboration. In this e-Learning system, the function of knowledge-point links can help students. Knowledge-points are organized in non-linear network structure, and are associated through the learning navigation. After the learners chose knowledge-point and the other requirements, and combined with user information, the system searches and organizes the related knowledge giving points' linked map, which is convenient for studying. This e-Learning system fits well in the problem-based learning model with knowledge transfer. It can fully stimulate the initiative of both teachers and students, and can support the teaching mode effectively.

2.8 Problem Design in Problem-based Learning

Designing problems for problem-based learning (PBL) courses in engineering has always been a challenging task, especially in environment where the only method of importing technical education has been through traditional a lecture/tutorial/practical approach. A Mantri at el. described design of problems, analyses of solutions submitted by the student groups and how learning objectives were achieved [22]. The facilitator not only managed time, but also keeps in mind what maximum technical nodes were covered and learning objectives were achieved. They described the flow of ideas touch the deep conceptual level and at the same time move to presentation levels. The pedagogy involved designing problems that covered the scope of the subject; carefully listing technical nodes and objectives; and handling the course, class, students and their psychological issues, besides the technical ones.

2.9 Group Effectiveness vs Individuals

Cooperative groups perform better than independent individuals on a wide range of problems. R. Laughlin at el. described the effectiveness of group size on intellective problems [20]. Comparisons of the performance of cooperative groups of a given size and individuals are a special case of the larger issue of the relationship between group size and performance. The current experiment addressed this larger issue by a comparison of groups of size two, three, four, and five people and the best of an equivalent number of individuals on letters-to-numbers problems. They review the surprisingly small amount of previous research on the effects of group size in problem solving. From these considerations, they predicted (a) better performance for groups of each of size two, three, four, and five than an equivalent number of individuals and (b) major improvement in performance from group size two to three, with decreasing improvement from group sizes three to four to five. Finally they suggest that 3-person groups are necessary and sufficient to perform better than the best individuals on highly intellective problems.

2.10 Constraints for Problem-based Learning

P. Lai *at el.* was study to obtain insight into the effect of quality assurance system on the implementation of PBL teaching strategy to courses [21]. Before the implementation of PBL, the tutors attended specific orientation and training sessions on PBL. Twenty-one tutors were randomly selected for interview after the implementation. The results of the study indicate that the quality assurance system within most institutions does affect the implementation of PBL. The reliance of research output and a standardized student feedback questionnaire as indicators of staff performance do have a detrimental effect on the implementation of PBL. On the other hand, resources and class size also have a direct effect on the willingness of academic staff to adopt the PBL approach in teaching. Finally, student factor also plays an important role in the successful implementation of PBL. To implement PBL successfully, PBL tutors need to be supported by a clear message from the university quality assurance system that this is the way to go forward. Without taking this seriously into consideration, one could predict serious difficulties in promoting PBL in the education sector.

2.11 Problem-based Learning Tools

2.11.1 Multiagents System for PBL

Fontes et al. introduced multiagents system to support problem-based learning [13]. Multiagents system can model complex system, allowing agents to have common or conflicting goals. According to this approach, four types of agents are proposed: a Problem Detector Agent (PDAg), a Student Agent (SAg), an Animated Interface Agent, and Work Group creation agents (WCAg). Those agents can interact with each other in two ways: directly (via communication and negotiation) or indirectly (acting upon the environment). The agents can cooperate in order to achieve mutual benefits or compete to serve their own interests. Sensors are the agent's data inputs and the actuators are the ways through which the agent per- forms its actions and interacts with the environment. Agents can perform many tasks in computer-supported collaborative learning, such as monitoring students' participation in discussions, facilitating the selection of topics for discussion, and assessing student performance in relation to the use of communication and cooperation tools available in the environment, among others. Fontes presented an approach that uses software agents to avoid allowing the students to lose focus during interactions with other students and support group creation, providing the facilitator with support to solve these problems. Using the proposed approach, it is possible to achieve a reduction of student dispersion, as upon detecting the focus has been lost, it notifies the facilitator, who can take appropriate action. The architecture also provides support for group creation. The WCAg is responsible for the automatic creation of groups by analyzing the students' profiles and the groups' profiles.

2.11.2 Web-based Environment for PBL

Yueh *et al.* developed web-based environment to implement problem-based learning [7]. Problem-based learning is a self directed learning method with different activities likes- team working, group discussion, collaboration and communication, resources sharing etc. Yueh developed this web-based environment to support all of those activities. This system contains of functions of general content management system such as announcement, course information, lecture notes of each class, and instructor's contacts. This system only describes the supporting features for problem-based learning. But it didn't mention about the problem distribution among the groups, problem complexity and others.

2.11.3 INDIE

INDIE was built to create web-based interactive learning environments where students can run simulated experiments, analyze test results, form rationales, and construct arguments to support or refute possible hypotheses by Lin Qiu [8]. Problem-based learning is a pedagogical strategy that centers learning activities around the investigation and development of solutions to complex and ill-structured authentic problems. A number of difficulties occur when implementing such approaches in schools. To address these difficulties Qiu developed this web-based tool. This paper focuses on how INDIE supports problem-based learning by creating an authentic environment that incorporates important aspects in real life, providing tools to help student perform problem-solving and receive coaching and critiquing, providing support for instructors to assess student understanding and provide feedback, and using an interface that allows open-ended inquiry and exploration. Problem-based learning environment also need more feature like- problem set generation, problem assignment and evaluation.

2.11.4 SQL-LES

SQL Learning and Evaluation System (SQL-LES) was developed by L. Hoque *et al.* [11]. SQL-LES focused on question bank, test set generation and evaluation of student's performance. The question bank used for both learning and automatic evaluation of student's performance by creating test set and assigning the test sets to the individual students. This system has been used for the teaching, learning and evaluation of database laboratory course in undergraduate level of the Department of Computer Science and Engineering (CSE), Bangladesh University of Engineering and Technology (BUET) in several years and found to be very effective in classroom environment. The overall SQL-LES architecture (Fig. 2.1) is a combination of six interconnected modules: Data Set Management Module (DSMM), User Management Module (UMM), System Security Module (SSM), Question Bank Management Module (QBMM), Test Set Management Module (TSMM) and Project Management Module (PMM).

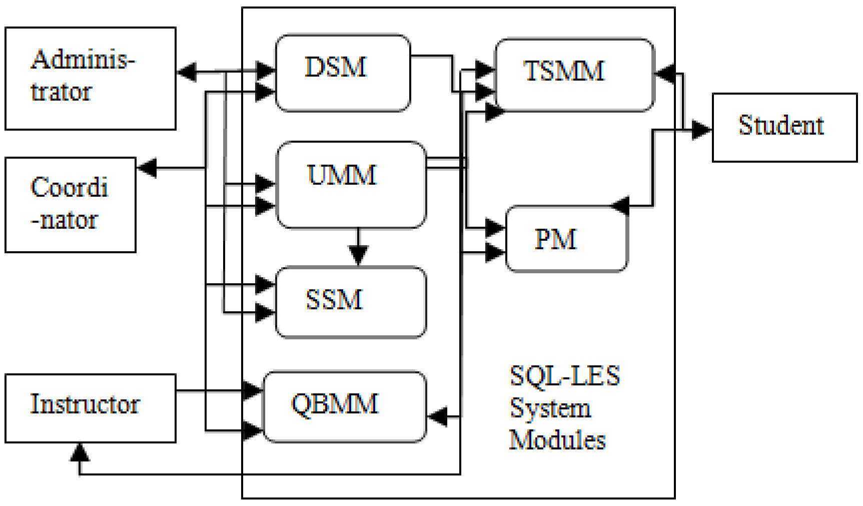

Fig. 2.1: System Architecture of SQL-LES

DSMM stores data related to schema given by system coordinator. This module is used by QBMM, TSMM or can be operated individually in response to authorized users. UMM is the hub of all user related functionality and interactions to other modules of system and are used by actors. It relies on SSM for authentication and authorization. SSM is the host of all security functionality of the system. It provides an abstract layer over other modules to protect them from external harm. QBMM deals with both executable and non-executable question managements and also schema management. Thus it has three sub-modules, Schema Management Sub-module, Executable Question Sub-module and Non-executable Question Sub-module. TSMM is responsible for creating and monitoring test sets. Test sets are built from schemas and questions supplied by QBMM. TSMM has two sub modules, one is for executable questions and another is for non executable questions. Project Management Module (PMM) helps students and instructors to submit and evaluate projects.

2.11.4.1 User Management Module (UMM)

In the system, there are 4 types of users: administrator, coordinator, instructor and student. UMM gets the authentication and authorization from SSM and send necessary information to TSMM, PMM and QBMM upon request. UMM can also register a user to the system and thus it can send and receive user information to or from SSM. Administrator can also modify a user and block or unblock a user. User can also access his full profile via UMM. The UMM has a function that fetch user auth info from SSM.

2.11.4.2 Data Set Management Module (DSMM)

SQL-LES contains preloaded datasets for creating new questions or update existing questions in the question bank. It contains a schema bank. Based on the schema bank, representative datasets are generated. It can also insert data to a table to be tested by student, can view current data and provide analysis and report on current data set, can export data as a CSV file (with help of UMM to determine the current user has the right to do the operation) and with support from SSM, can provide additional security over DSMM.

2.11.4.3 Test Set Management Module (TSMM)

Using these module teachers can create test sets for student examination on SQL, assessment and practice. This module depends on QBMM module for questions. It fetches questions from QBMM and generates test sets. A test set can be assigned to every individual student in the class; time can be set and managed. Instructors can monitor every submission of the students. The instructor can download the total class performance of all the students and give to the students just after the class is over. Sometime, the students claim that their submission was correct but the system has evaluated wrongly.

This module deals with 2 types of test sets. Namely, Executable test set and Non executable test set. Executable test sets deals with executable questions. An executable question can be evaluated automatically by the system. As for example, SQL questions are executable type. TSMM deals with question setup, test set creation, test monitoring and test result. When a student log in into the system, he finds the test set assigned to him by the instructor. The student can view the question, the SQL schema with data types and the relational schema. He can perform all kinds of checking whether his solution to the SQL problem given to him is correct or not. After all kinds of checking, he submits the solution and gets an instant response whether his given answer is correct or not.

2.11.4.4 Question Bank Management Module (QBMM)

A question bank is the storage of questions related to a topic (in this SQL), where these questions are related to a schema stored in the schema bank of the system (Fig.6). QBMM module stores the schema, related Entity Relation Diagram (ERD)'s, and relations in system. This module serves other modules and used by instructor to create question sets based on a specific schema. Instructors can assign for each question a complexity value that is used for test set creation.

The QBMM module has the two sub-modules, SQL executable module and non-executable module. The purpose of SQL executable module is to deal with those questions which can be executed by the system. In this case, system can fully determine that the learner has given a correct solution or not. The executable module has two sub-modules, Schema Bank and SQL executable question bank.

Non executable module serves those questions that cannot be solved automatically, Such as database design, PL-SQL functions and procedure evaluation. In this case, these solutions have to be checked manually and evaluated by instructor. In general, QBMM deals with schema and questions that are put to test the learning.

2.11.4.5 System Security Module (SSM)

It is an internal module that works throughout the system to provide enhanced security on data and programs executed inside the system. SSM has user security layer which works with UMM, project security layer which works with PMM and SQL security layer that works when there is a SQL execution on system or when an SQL related information is storing in system. This sub module checks the SQL statement and confirms that this SQL statement will not harm the system.

For data sets security, SMM applies SQL security level while communicating with data storage (either database or configuration files required for a DSMM operation) and it help building improved security with Database server (in this case as we are using oracle security services) by adding some upper layer functionality for developing intelligence data security.

2.11.4.6 Project Management Module (PMM)

PMM module deals with managing the projects throughout a semester. Its sole purpose is to automate the project activity management and communicate with other modules if necessary.

This module is used by admin, instructors and students. Admin supervise the whole process while instructors and students communicate and transfer different state in this module. State refers to submission of project designs, reports and presentation during the tenure of the project.

A project requires a strong evaluation process during a course like database system design. A project demonstrates how a student can handle a real life problem with the knowledge he learned from the course. Evaluation of project is a difficult task as there are many process and artifacts to control. The PMM helps an instructor to handle these tasks. The necessary steps taken by the PMM Module are commencing project session, submissions of project list, creating project group by student, assignment of project supervisor, assign projects to group, project submissions, and Project evaluation.

The present system evaluates the SQL queries as correct or incorrect. No partial evaluation can be done using the system. Also there is no global complexity value of SQL problems in the present system. Teachers assign the complexity value during the creation of question bank trough QBMM. Analyzing of the SQL queries and the results, a model can be developed for assigning a global complexity value of the SQL problems. The same model can be used for partial evaluation of submitted SQL solutions.

2.11.5 ShareFast

Kazuo et al. developed a new design engineering educational framework using an e-learning system called ShareFast, a Semantic Web-based software for document management system with workflow [13]. The software offers a function to keep tracks of learner's behavior so that the instructor can analyze it to improve learning materials and class efficiency. It can also record learner's input and output history data for the instructor to conduct performance analysis activities. This tool has been developed by the members of Design Engineering Laboratory, the University of Tokyo. It is an open source, client/server application for document management based on workflow using RDF metadata on Jena framework. The client program, developed using C# technology, provides a workflow editor for users to create workflows and relate any documents to each task node in the workflows. It also provides the Tree Explorer to browse the workflows hierarchically. Workflows will be stored as XML with their metadata (e.g. creator, create date etc.) in RDF format. The client program uploads them together to the server. ShareFast has been applied for many aspects, such as knowledge management, information sharing environment and design support system. However, one of the very first reasons to develop ShareFast is to use it as the e-learning component of a framework for supporting design education, such as CAD software learning. ShareFast system contains many functions and activities, which can powerfully facilitate teacher and students in design learning activities.

2.11.6 Problem-based Learning via Web

Eleni *at el.* developed web base learning environment to support problem-based learning [23]. In this approach, students and instructors use the web as a virtual place to collaborate and create new knowledge and new educational experiences. Specific objectives of this work include- support collaboration of remote overspecialized medical experts in order to devise, develop and deploy didactic problems for problem based learning in medicine; deploy problem-based sessions in virtual teams, where both students and instructors may be located in remote institutions; support strong instructor's presence; provide tools for student inquiry and collaboration; and provide mechanisms for continuous monitoring and evaluation, that would address direct knowledge, as well as tacit competencies targeted via PBL. Considering the academic educational set-up, there is also the additional requirement for integration with generic environments that support teaching in higher education, i.e. open source learning management systems and related educational standards. This approach combines collaborative tools such as wikis, blogs and forums in order to provide problem based learning solely on the web. In these PBL sessions, instruction is performed by an interdisciplinary team of experts from remote institutions, while the group of learners can be students from the same or different institutions within the consortium. Instructors collaboratively develop a problem in a wiki. Discussion is initiated via a problem's blog or forum, where students and instructors collaborate to analyze the problem, identify conquered knowledge and plan actions for problem solving. Then students search (via the web and not only) and collaborate to solve the case via the wiki. Student activities, progress and more importantly gained experience and competences are recorded, shared and commended on via their personal blogs. The entire learning episode and all its steps (with the final problem/answer deployment) are recorded, commended on and monitored via the wiki (final and intermediate versions) and the participants' blogs.

2.9 Summary

This chapter described different facets of existing problem-based learning systems. Existing PBL systems focused on the PBL methodology, teaching strategy, problem design, group distribution, teacher and student roles, learning environment, applicable educational sectors and limitation of PBL. However, PBL is a highly successful model for teaching and learning. Still it has been required lots of research to make it more effective and efficient. Complexity model is a new technique to find out problem complexity. The existing PBL researches do not focus on the distribution of similar types of problem distribution among the student groups. Complexity model will help to find out analogous problem using problem complexity value.

In the next chapter we have discussed system architecture and analysis of our developed complexity model for problem-based learning.

Chapter 3
Complexity Model: System Architecture and Analysis

Problem-based Learning (PBL) is a blended learning environment, a combination of self-directed learning and collaborative learning. This model believes that teaching should not only directly focus on the knowledge of the subjects, but also focus on the learner's abilities, such as the ability to analyse and solve problems, communicating skills and comprehensive ability. Problem is the core element in Problem-based Learning. By solving the practical problems, the learners can explore the concept and principles behind the issues, developing their self-learning ability, and implement the meaningful construction of the knowledge. In PBL, different levels of problems are distributed among the student to solve. It is important to distributed similar level of problem between different students. We have determined the problem level depend on complexity value of a problem. Problem complexity depends on how much domain knowledge and general knowledge requires to solve the problem. A Structured Query Language (SQL) statement is a combination of database clauses. We have analysed SQL statement in top-down method to know details about database clause and use position. We have marked each clause with complexity value. We have calculated complexity value of each clause to find out the complexity value of a SQL problem. Finally, we have found out the equivalence of problems using complexity value of different problems. We have taken several surveys from database specialist about the complexity value of individual database clause based on the clause type and use position in SQL statement.

3.1 Complexity Model

Problem complexity depends on how much domain knowledge requires to solve a problem. To find out used domain knowledge, Complexity Model parses problem in to sub-problem. Then each sub-problem analyzes to find out critical, meaningful and complex item to solve the problem. Then we assign complexity value to each item based on item type and use position in SQL statement. We have used top-down method to analyze PBL problem.

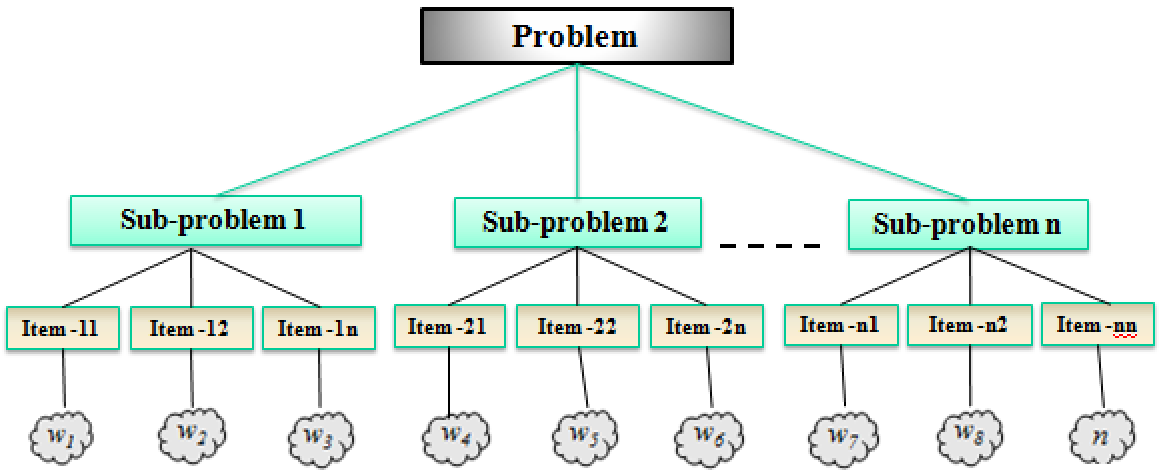

Fig. 3.1: Top-down Analysis of PBL Problem

3.2 Complexity Value

Data is an important factor in any programming concept. Database operations are key items in any programming language

$$\text{Complexity Value}(C_P) = \sum_{i=1}^{n} (w_{ti} \times \log_2(1+k))$$

where,

n = number of sub-problem

k = number of similar item in sub-problem

t = item type

w_i = complexity value of *t* type item

Database SQL statement is a combination of different SQL clause, function, predicate, constraint and others. Same item can use in different place or repeat again and again within SQL statement which does not seem to increase the complexity of SQL statement. To curb complex value, we have used logarithm function.

3.3 Equivalence of Problem

In PBL, it is necessary to assign similar level of complex problems among the students. Teacher uses complexity value of a problem to determine problem equivalence. The equivalence of problems means that the complexity to solve the problem is within a specified boundary. Those problems fall into a specified boundary are all equivalent problems and any of the problem can be assigned to any student and seem to have equal judgement. The boundary value depends on the problem domain and teaching policy. During problem equivalence, the domain of a problem must also be considered. Let us consider domains D_1, D_2 ... D_m are in an area of problems and P_1, P_2 ... P_n are problems in domain D_1 with complexities C_1, C_2 ... Cn. The problem P_1 will be equivalent to P_2 if

$$C1 \sim C2 \leq \text{€}$$

where, **€** is the allowable error in the specific level of problem.

The value of **€** can vary based on the nature of problems, how many levels we want to divide the problem for a particular domain and what is the difference between the lowest and the highest complexity value. For simple level of complex problems, minimum domain knowledge requires to solve SQL problems and the difference between the lowest and the highest complexity value is minimum. In this case, we have chosen minimum value for €. Otherwise all problems will be in same level. In the following table (Table 3.1) has shown the problem level using boundary value.

Table 3.1: Equivalence of Problems using Boundary Value

Problem No	**Complexity Value (C_P)**	**Problem Level**	**Boundary Value (CP ± €)**
1	23	Level – 1	25 ± 2
2	23		
3	26		
4	29	Level – 2	30 ± 2
5	30		
6	28		

Here we have divided the problem in two levels. If we want to divide those problems in more levels then we have to decrease the value of **€**.

Rule: Two problems can be considered equivalent only and only if they are in the same domain and the complexity value is within a specific boundary.

3.4 Basic SQL Operation

Data is an important factor in any programming concept. Database operations are key items in any programming language. In general, all database operations can be broadly classified into the following categories

- SELECT
- CREATE

- INSERT
- UPDATE
- DELETE

3.4.1 Select Statement

SQL is a special purpose programming language to manipulate data in Relational Database Management System. The most common operation in SQL is the query, which is performed with the declarative SELECT statement. SELECT retrieves data from one or more tables, or expressions. Select statement has following clauses with huge selection of options, parameters and keyword:

- **FROM CLAUSE** -indicate data source from which data to be retrieved
- **WHERE CLAUSE** -uses to specify which data to be retrieved
- **GROUP BY CLAUSE** -groups data to apply aggregate function
- **HAVING CLAUSE** -uses with Group By clause to filter groups
- **ORDER BY CLAUSE** -identifies which columns are used to sort the resulting data

3.4.1.1 General Format of SQL SELECT Statement

```
SELECT [ALL | DISTINCT] column1[,column2] FROM table1[,table2 | Sub Query]
[WHERE "conditions"] [GROUP BY "column-list"] [HAVING "conditions”] [ORDER
BY "column-list" [ASC | DESC]]
```

Complexity of a given problem depends on how many database clauses has used with options and parameters.

3.4.1.2 SELECT Clause

Select clause itself can use predicate, function and expression like-

```
SELECT [ALL | DISTINCT | TOP n] column1[,column2]
[function(column1[,column2])] [expression]
```

In SQL, there are two types of function. First one is library or built in function and second one is user defined function. Function complexity depends on which type of function is used, how many parameters are required for that function and how to use those parameters to execute that function. Select statement uses Expression to format, represent and calculate column value.

3.4.1.3 FROM Clause

Select statement uses FROM as required clause to define the data source as-

```
FROM table1[,table2 | Sub-Query].
```

Select statement can include optional Sub-Query with FROM clause as data source. The complexity of SELECT statement increases with the number of used table and sub-query. To link with different table JOIN keyword can use with linking condition. The syntax of JOIN keyword like-

```
From tabl1 t1 <JOIN> table2 t2 on t1.coulmnA = t2.columnA
```

3.4.1.4 WHERE Clause

WHERE clause in SQL statement specifies that query should only affect rows that meet specified criteria. The criteria are expressed in the form of predicate or condition. Where clause is not mandatory clause, but can be used to limit the number of affected rows. The syntax of SQL WHERE clause:

```
WHERE comparison predicates [, other predicates]
```

3.4.1.5 GROUP BY Clause

The SQL GROUP BY Clause is used along with the aggregate functions to retrieve data grouped according to one or more columns. The common format of GROUP BY clause is –

```
SELECT column1, aggregate-function(column2)FROM table1[,table2 |
Sub Query] GROUP BY column1[,column2]
```

3.4.1.6 HAVING Clause

HAVING clause in SQL specifies that an SQL SELECT statement should only return rows where aggregate values meet the specified conditions. It was added to the SQL language because the WHERE keyword could not be used with aggregate functions. The syntax of HAVING clause is:

```
HAVING comparison predicates [, other predicates]
```

3.4.1.7 ORDER BY Clause

The ORDER BY clause is an optional clause in SQL SELECT Statement and use to sort the resulting data, and in which direction they should be sorted. The syntax of order by clause is:

```
ORDER BY column1[,column2] [ ASC | DESC ]
```

3.4.2 CREATE Statement

A CREATE statement in SQL creates an object inside of a relational database management system (RDBMS). The types of objects that can be created depends on which RDBMS is being used, but most support the creation of tables, indexes, users, synonyms and databases. A commonly used CREATE command is the CREATE TABLE command. The typical usage is:

```
CREATE [TEMPORARY] TABLE [table name] ( [column definitions] ) [table
parameters].
```

Column definitions: A comma-separated list consisting of any of the following

Column definition: [column name] [data type] {NULL | NOT NULL} {column options}

Primary key definition: PRIMARY KEY ([comma separated column list])

Constraints: {CONSTRAINT} [constraint definition]

3.4.3 INSERT Statement

An SQL INSERT statement adds one or more records to any single table in a relational database. Insert statements have the following form:

```
INSERT INTO table (column1 [, column2, column3 ... ]) VALUES (value1 [,
value2, value3 ... ])
```

The number of columns and values must be the same. If a column is not specified, the default value for the column is used. The values specified (or implied) by the INSERT statement must satisfy all the applicable constraints (such as primary keys, CHECK constraints, and NOT NULL constraints). If a syntax error occurs or if any constraints are violated, the new row is not added to the table and an error returned instead.

3.4.4 UPDATE Statement

An SQL INSERT statement adds one or more records to any single table in a relational database. An SQL UPDATE statement changes the data of one or more records in a table. Either all the rows can be updated, or a subset may be chosen using a condition. The UPDATE statement has the following form:

```
UPDATE table_name SET column_name = value [, column_name = value ...]
[WHERE condition]
```

For the UPDATE to be successful the user must have data manipulation privileges (UPDATE privilege) on the table or column and the updated value must not conflict with all the applicable constraints (such as primary keys, unique indexes, CHECK constraints, and NOT NULL constraints).

3.4.5 DETELE Statement

In the database structured query language (SQL), the DELETE statement removes one or more records from a table. A subset may be defined for deletion using a condition, otherwise all records are removed. The `DELETE` statement follows the syntax:

```
DELETE FROM table_name [WHERE condition];
```

Any rows that match the WHERE condition will be removed from the table. If the WHERE clause is omitted, all rows in the table are removed. The DELETE statement should thus be used with caution. The DELETE statement does not return any rows; that is, it will not generate a result set. Executing a DELETE statement can cause triggers to run that can cause deletes in other tables. For example, if two tables are linked by a foreign key and rows in the referenced table are deleted, then it is common that rows in the referencing table would also have to be deleted to maintain referential integrity.

3.5 Top-down Analysis of SQL Select Statement

SQL SELECT statement is a combination of different database clause named FROM, WHERE, GROUP BY, HAVING and ORDER BY. FROM clause uses as a required clause and all other clauses are optional. Each clause can use functions, predicates, columns or expression to make it meaningful. The main purpose of database clauses is to prepare data for user with desire shape.

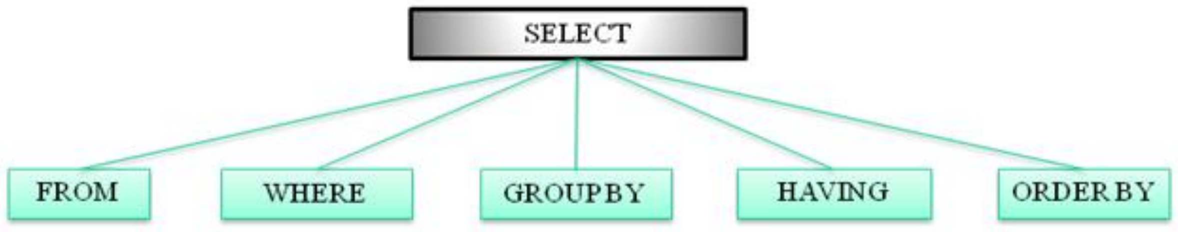

Fig. 3.2: Top-down Analysis of Select Statement

3.5.1 Tree Structure of SQL Select Statement

Database clauses can use lots of keyword, function, expression and predicate to make the SQL query more purposeful and efficient. Any clause can use those items with proper format. The level of used of those items have shown using tree structure.

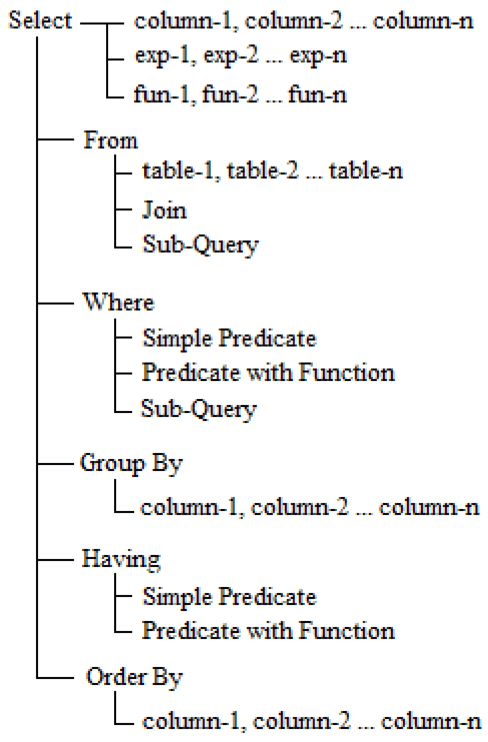

Fig. 3.3: Tree Structure of SQL Select Statement

3.5.2 Level of Used of Database Clauses

Select statement has five clauses. Each clause can use function, column name, predicate or sub-query as parameter. Same type parameter can use with different clauses and there is different meaning. So the complexity of the used parameter depends on the type and use position in SQL statement. In the following figure (Fig. 3.4), we have shown details about use position of different parameter with different clauses.

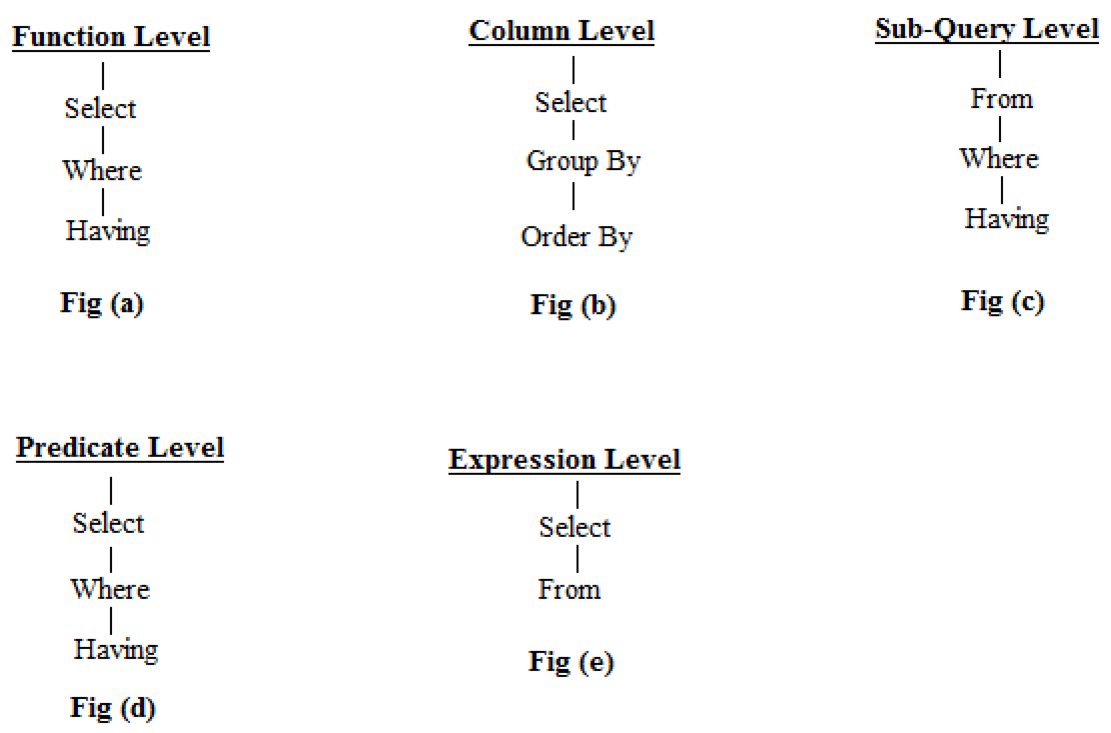

Fig. 3.4: Used Level of SQL Database Clauses

3.5.3 Complexity Value of Database Clauses with SELECT Statement

Table 3.2: Complexity Value of SQL Items with Different Clause

Name of the SQL Clause	Used in Level	Complexity Value
Functions	Select	*fw*
	Where	
	Having	
Columns	Select	C*v*
	Group By	
	Order By	
Tables	From	*Tv*
Predicate	Select	P*v*
	Where	
	Having	
Expression	Select	E*v*
	From	

3.5.4 Complexity Value of SQL SELECT Operation (C_P)

Complexity value of a given SQL problem has been calculated by analyzing SQL statement. SQL statement is a combination of different types of clause. Database clauses can use lots of keyword, function, expression and predicate to make the SQL query more purposeful and efficient. So SQL complexity depends on used clauses and parameters. The complexity value has been calculated by using the following formula:

$$C_P = \left(\left(\sum_{i=1}^{n} \frac{C_i}{C_{i(max)}}\right)/n_{max}\right) \times 100, \quad C_i \in \{CL_v\,,\ F_v\,, C_v\,, T_v\,, P_v\,, E_v\}$$

where

n_{max}	is the cardinality of the clause array of C_i
CL_v	is complexity value of usages database clauses
F_v	is functional value,
C_v	is columns value,
T_v	is tables value,
P_v	is predicates value,
E_v	is expression value

3.5.4.1 Complexity Value of Function (F_v)

The complexity SQL function depends on which type function uses to process data, how many parameters requires for that function and which clause use that function. Complexity vale of SQL Select statement proportionally increased with the functional value. To calculate the complexity value of usage function, we have used the following formula:

$$F_v = \sum_{t=0}^{m} f_{wt} \times \log_2(1+n)$$

where,

t	is function type,
f_w	is the functional weight of type t,
n	is the number of used t type function

Same type of function can repeat in SQL statement again and again, which does not mean to increase the problem complexity. To curb the complexity value for repeated function, we have used logarithm function.

SQL Functions:

Table 3.3: Database Function with Type and Weight

Function Name	Function Type (t)	Function Weight (f_w)
SUM	t	f_w
COUNT	t	f_w

AVG	t	f_w
MAX	t	f_w
MIN	t	f_w
FORMAT	t	f_w
etc		

Example:

If we consider the functional weight (f_w) of Type t is 1, then the functional value will be –

Table 3.4: Complexity Value of Used Function

Number of used function	**Function value** $\boldsymbol{F_v = f_w \times \log_2(1+n)}$
1	$1 \times \log_2(1+1) \approx 1$
2	$1 \times \log_2(1+2) \approx 1.585$
3	$1 \times \log_2(1+3) \approx 2$
4	$1 \times \log_2(1+4) \approx 2.322$
5	$1 \times \log_2(1+5) \approx 2.585$

3.5.4.2 Complexity Value of Table (T_v)

Table is a set of data elements (values) that use as a data source with SQL query. Multiple tables can use with SQL query to retrieve user desire data. JOIN clause use to combine data from different tables. User needs to carefully handle table joining to avoid wrong data. So query complexity will be increase if we use more tables to get data. We have the following formula to calculate used table value:

$$T_v = t_w \times \log_2(1+n)$$

where,

t_w is the weight of table

n is the number of used tables

Multiple tables can use in SQL statement, which does not seem to increase the problem complexity. To curb the complexity value for multiple tables, we have used logarithm function.

Complexity Value by Varying the Number of Table:

Table 3.5: Complexity Value of Used Table by Varying Number of Tables

No. of Table	**Complexity Value**

1	$5 \times \log_2(1+1) \approx 5$
2	$5 \times \log_2(1+2) \approx 7.925$
3	$5 \times \log_2(1+3) \approx 10$
4	$5 \times \log_2(1+4) \approx 11.61$
5	$5 \times \log_2(1+5) \approx 12.925$
6	$5 \times \log_2(1+6) \approx 14.035$
7	$5 \times \log_2(1+7) \approx 15$
8	$5 \times \log_2(1+8) \approx 15.85$
9	$5 \times \log_2(1+9) \approx 16.61$
10	$5 \times \log_2(1+10) \approx 17.295$
17	$5 \times \log_2(1+17) \approx 20.85$

3.5.4.3 Complexity Value of Column (C_v)

Table has a specified number of columns. Most of the time user does not need all column value. In this situation, user has to mention desired column in SQL query. Also some aggregate function depends on column which has to define by user. So column has many roles in QSL query. Column value (C_v), depends on the number of used column and the level of used. We have calculated column value in the following way:

$$C_v = c_{vS} \times \log_2(1+m) + c_{vG} \times \log_2(1+n) + c_{vO} \times \log_2(1+p)$$

where,

S	is SELECT clause,
O	is other clause like GROUP BY, ORDER BY,
c_{wS},	is the column weight with SELECT clause,
c_{wG},	is the column weight with GROUP BY clause,
c_{wO},	is the column weight with ORDER BY clause,
m	is the number of used column with SELECT clause
n	is the number of used column with GROUP BY and
p	is the number of used column with ORDER BY clause

Table 3.6: Complexity Value of Column with Different Clauses

Number of Column	Used with	Column Weight	Complexity Value
2	SELECT	1	1.585
4			2.322
6			2.807
2			4.755

4	GROUP BY/ ORDER BY	2	6.966
6			8.421

3.5.4.4 Complexity Value of Predicate (P_v)

Predicates boil down to either a TRUE or a FALSE result. Predicates use to filter out unwanted rows from the result of an SQL query by applying a WHERE clause whose predicate excludes the unwanted rows.

$P_v = \sum_{t=0}^{n} p_{wt} \times \log_2(1+n)$

where

t	is the predicate type
p_w	is the weight of *t* type predicate
n	is the number of use *t* type predicates

Comparison Predicates:

Table 3.7: List of Comparison Type Predicates

Predicate	Meaning
=	Equal
<>	Not equal
<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal

Other Predicates:

Table 3.8: List of Predicates

ALL	BETWEEN
DISTINCT	EXISTS
IN	LIKE
MATCH	NOT IN
NOT LIKE	NULL
OVERLAPS	SIMILAR
SOME, ANY	UNIQUE
TOP	SKIP
etc	

3.5.4.5 Complexity Value of Expression (E_v)

SQL Expression is a combination of symbols and operators to perform arithmetic calculation, formation and compare values against others value. Expressions can be found inside of any SQL clause usually in the form of a conditional statement.

$$E_v = \sum_{t=0}^{m} e_{wt} \times \log_2(1+n)$$

where,

t	is the expression type,
e_w	is the expression weight of type t,
n	is the number of used t type function

3.6 Algorithm to Calculate Complexity Value of SQL problem

Algorithm I calculates complexity value of a given SQL problem using SQL statement for that problem. This algorithm uses five functions - CalculateFunctionalValue, CalculateColumnValue, CalculateTableValue, CalculatePredicateValue and CalculateExpressionValue to complete the calculation. We have described details about those functions later in this chapter.

Algorithm I ComplexityValue

```
Input: SQL Statement sst;
Output: return total Problem weight as Complexity value in numeric
Step 1: Initialize Total Weight totalWeight = 0.0;
Step 2: totalWeight += CalculateFunctionalValue(sst);
Step 3: totalWeight += CalculateColumnValue(sst);
Step 4: totalWeight += CalculateTableValue(sst);
Step 5: totalWeight += CalculatePredicateValue(sst);
Step 6: totalWeight += CalculateExpressionValue(sst);
```

3.6.1 Algorithm to Calculate Complexity Value of Function

SQL statement uses function to make query result more meaningful to user. Function can use anywhere in SQL query with desired parameter(s). Function complexity depends on which type of function is used, how many parameters require for that function and how to uses those parameters to execute that function. Functional weight has been calculated by using the Algorithm II.

Algorithm II CalculateFunctionalValue

```
Input: SQL Statement sst;
Output: return functional weight in numeric
Step 1: Initialize functional weight fnWeight = 0.0, Function list with Weight fnList[];
        usedFnByType[ftype] = 0.0; to count similar type function.
Step 2: ARRAY[] items = split sst to words;
Step 3: FOREACH(item in items)
                IF( item in fnList) THEN
                        ftype = find out function type
                        usedFnByType[ftype] +=1;
                ENDIF
        END FOREACH
Step 4: FOREACH ( usedFn in usedFnByType)
                fnWeight = fnList[ftype] * log(1+ usedFn, 2);
        END FOREACH
```

In the Algorithm II, we have calculated functional complexity of a SQL statement. First we have split SQL statement into array, and then we have searched the array to find out the use database function. Finally we have calculated complexity value of function based on their type. The time complexity of step 1 is O(1), step 2 is O(1), step 3 for find out used function from *n* number of items is O(n) and step 4 for calculate complexity value of m number of function is O(m). The total complexity for Algorithm II is

$= O(1 + 1 + n + m)$

$= O(2 + n + m)$

$= O(n + m)$

The complexity of Algorithm II for calculation of complexity value for function is $O(n + m)$ where n is the number of used item in SQL statement.

3.6.2 Algorithm to Calculate Complexity Value of Column

Database table has a specified number of columns. Most of the time user needs to mention column name at different level in SQL query to perform calculation, data filtering, ordering and others. Algorithm III has been used to measure the functional weight.

Algorithm III CalculateColumnValue

```
Input: SQL Statement sst;
Output: return column weight as numeric value
Step 1: Initialize column weight cWeight = 0.0; Resrver SQL keyword keyList[];
        usedColumnByLevel[usedLevel] = 0.0; to count total used column at different level
Step 2: ARRAY[] items = split sst to words;
Step 3: FOREACH(item IN items)
                IF( item NOT IN keyList) THEN
                        usedLevel = find out level [Select, Group By or Order By]
                        usedFnByType[usedLevel] +=1;
                ENDIF
        END FOREACH
Step 4: cWeight = ColumnWeightWithSelect * log(1+ usedFnByType[0], 2);
        cWeight += ColumnWeightWithOthers * log(1+ usedFnByType[1], 2);
```

In the Algorithm III, we have calculated the complexity value of columns of a SQL statement. First we have split SQL statement into array, and then we have searched the array to find out the use database column and the clause name which use that column. Finally we have calculated complexity value of column based on use position. The time complexity of step 1 is O(1), step 2 is O(1), step 3 for find out used function from *n* number of items is O(n) and step 4 is O(1). The total complexity for Algorithm II is

$= O(1 + 1 + n + 1)$

$= O(3 + n)$

$= O(n)$

The complexity of Algorithm III for calculation of complexity value for column is $O(n)$ where n is the number of used item in SQL statement.

3.6.3 Algorithm to Calculate Complexity Value of Table

SQL statement uses function to make query result more meaningful to user. Function can use anywhere in SQL query with desired parameter(s). Function complexity depends on which type of function is used, how many parameters require for that function and how to uses those parameters to execute that function. Functional weight has been calculated by using the Algorithm IV.

Algorithm IV CalculateTableValue

```
Input: SQL Statement sst;
Output: return table weight as numeric value
Step 1: Initialize Total Weight tWeight = 0.0;  used table uTable = 0.0
Step 2: uTable = Total used table in SQL statement
Step 3: tWeight = tableWeight * log(1+ uTable, 2);
```

3.6.4 Algorithm for Complexity Value of Predicate

SQL statement uses function to make query result more meaningful to user. Function can use anywhere in SQL query with desired parameter(s). Function complexity depends on which type of function is used, how many parameters require for that function and how to uses those parameters to execute that function. Functional weight has been calculated by using the Algorithm V.

Algorithm V CalculatePredicateValue

```
Input: SQL Statement sst;
Output: return functional weight in numeric
Step 1: Initialize predicate weight pWeight = 0.0, Predicate list with Weight pList[];
        usedFnByType[], to store used predicate
Step 2: ARRAY[] items = split sst to words;
Step 3: FOREACH(item in items)
                IF( item in fnList) THEN
                        usedFnByType[i++] = item;
                ENDIF
        END FOREACH
```

```
Step 4: FOREACH ( usedFn in usedFnByType)
                fnWeight = fnList[ftype];
        END FOREACH
```

In the Algorithm V, we have calculated the complexity value of predicates of a SQL statement. First we have split SQL statement into array, and then we have searched the array to find out the use database predicate with predicate type. Finally we have calculated complexity value of predicate based on their type. The time complexity of step 1 is O(1), step 2 is O(1), step 3 for find out used predicate from *n* number of items is O(n) and step 4 for calculate complexity value of m number of predicate is O(m). The total complexity for Algorithm V is

$= O(1 + 1 + n + m)$

$= O(2 + n + m)$

$= O(n + m)$

The complexity of Algorithm V for calculation of complexity value for predicate is $O(n + m)$ where n is the number of used item in SQL statement.

3.6.5 Algorithm to Calculate Complexity Value of Expression

SQL statement uses function to make query result more meaningful to user. Function can use anywhere in SQL query with desired parameter(s). Function complexity depends on which type of function is used, how many parameters require for that function and how to uses those parameters to execute that function. Functional weight has been calculated by using the Algorithm VI.

Algorithm VI CalculateExpressionValue

```
Input: SQL Statement sst;
Output: return Expression weight as numeric value
Step 1: Initialize Expression weight ExWeight = 0.0; Resrver SQL keyword keyList[];
        usedColumnByLevel[usedLevel] = 0.0; to count total used column at different level
Step 2: ARRAY[] items = split sst to words;
Step 3: FOREACH(item IN items)
                IF( item NOT IN keyList) THEN
                        extype = find out expression type
                        usedExByType[extype] +=1;
```

```
          ENDIF
     END FOREACH
Step 4: FOREACH ( usedEx in usedExByType)
          ExWeight = fnList[usedEx] * log(1+ usedEx, 2);
     END FOREACH
```

In the Algorithm VI, we have calculated the complexity value of expression of a SQL statement. First we have split SQL statement into array, and then we have searched the array to find out the use expression with type. Finally we have calculated complexity value of expression based on their type. The time complexity of step 1 is O(1), step 2 is O(1), step 3 for find out used expression from n number of items is O(n) and step 4 for calculate complexity value of m number of expression is O(m). The total complexity for Algorithm VI is

$= O(1 + 1 + n + m)$

$= O(2 + n + m)$

$= O(n + m)$

The complexity of Algorithm VI for calculation of complexity value for function is $O(n + m)$ where n is the number of used item in SQL statement.

3.7 Top-down Analysis of SQL Create Table Statement

Create Table statement in SQL, creates a table object in relational database. This statement is a combination of Column Definition and Constraint. Constraint imposes some rules on table or columns using default, unique, check etc.

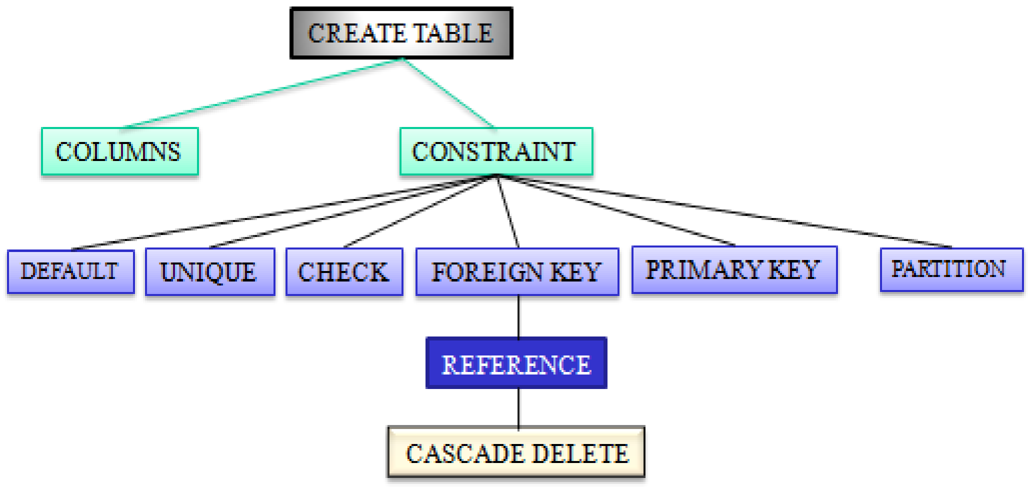

Fig. 3.5: Top-down Analysis of Crete Statement

3.7.1 Tree Structure of SQL Create Statement

SQL Create statement uses different constraints to describe the desired behavior of column value. We have shown the level of used of different constraints in bellow figure using tree structure.

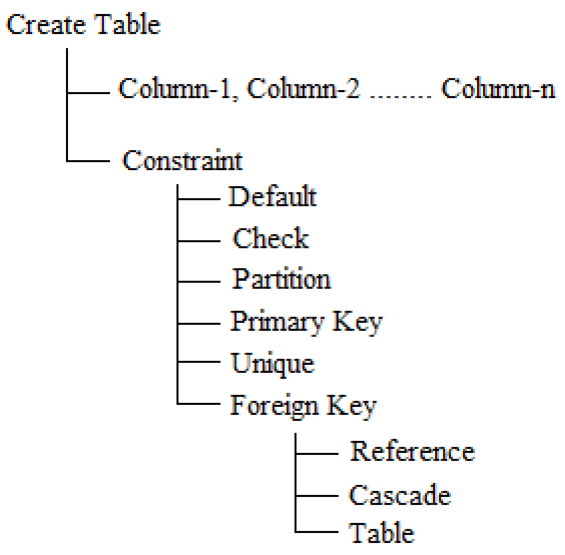

Fig. 3.6: Tree Structure of Crete Statement

3.7.2 Complexity Value of Database Clauses with CREATE Statement

Table 3.9: Complexity Value of Different Clauses for Create Statement

Name of the SQL Clause	Used in Level	Complexity Value
Create Tables		*CTv*
Column		C_v
Table	Constraint	t_w
Constraint		K_w

3.7.2 Complexity Value of SQL CREATE Statement (C_P)

SQL functional value depends on which type function used to process data, how many parameter requires for that function, which clause use that function etc. SQL complexity proportionally increased with the functional value. To calculate functional value we have used the following formula:

$$C_P = CT_v + C_v \times \log_2(1+p) + \sum_{t=0}^{m} K_{wt} \times \log_2(1+n) + t_w \times \log_2(1+q)$$

where,

CT_v	is the weight of CREATE Table command
C_v	is the weight of column
p	is the number of used columns
K_w	is the weight of constraint
t	is the type of used constraint
n	is the number of used t type constraints
t_w	is the weight of table
q	is the number of used tables

3.8 Top-down Analysis of SQL Insert Statement

SQL Insert statement is a combination of column name and column value use to insert data in data table.

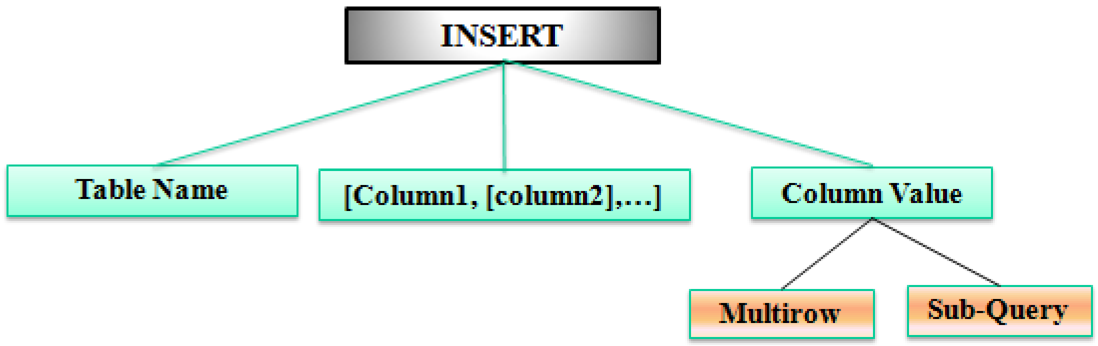

Fig. 3.7: Top-down Analysis of SQL Insert Statement

3.8.1 Tree Structure of SQL Insert Statement

SQL Insert statement use column name as optional item. Using column name, it is possible to insert particular column value. The level of use column name, column value and sub-query has shown in bellow figure using tree structure.

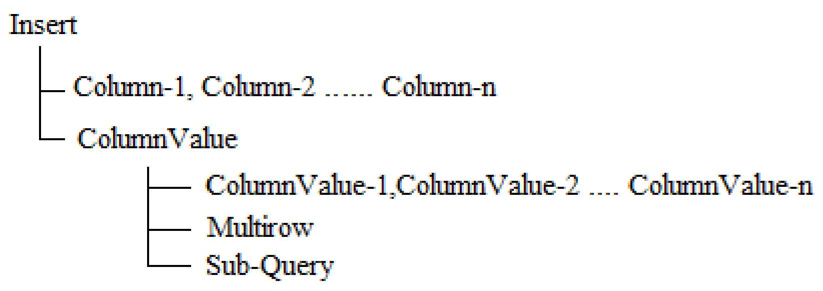
Insert
Column-1, Column-2 ...... Column-n
ColumnValue
ColumnValue-1,ColumnValue-2 .... ColumnValue-n
Multirow
Sub-Query

Fig. 3.8: Tree Structure of SQL Insert Statement

3.8.2 Complexity Value of Database Clauses with Insert Statement

To calculate complexity value of Insert statement, we have used the following table.

Table 3.10: Complexity Value of Different Clauses with Insert Statement

Name of the SQL Clause	Complexity Value
Insert	*Iv*
Column	C_w
Column Value	C_v
Multirow	M_v
Sub-Query	*

3.8.3 Complexity Value of SQL Insert Statement (C_P)

Complexity value of Insert statement depends on the number of mention column name, column value and number of rows insert at a time. To calculate complexity value we have used the following formula:

$$C_P = I_v + C_w \times \log_2(1+p) + C_v \times \log_2(1+n) + M_v \times \log_2(1+q)$$

where,

I_v	is the weight of INSERT command
C_w	is the weight of column
p	is the number of used columns
C_v	is the weight of column value
n	is the number of column value
M_v	is the weight of multi row
q	is the number of rows insert at a time

Same item can repeat again and again. To limit the complexity value, we have used logarithm function.

3.9 Top-down Analysis of SQL Update Statement

SQL Update statement uses to change particular column value in a data row. Update statement uses column name and column value with update condition. Update condition can use as optional parameter. Without update condition, update statement will update entire table value for the mention column.

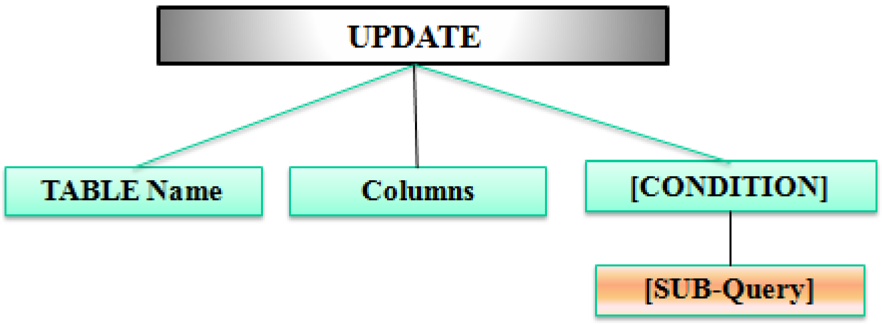

Fig. 3.9: Top-down Analysis of SQL Update Statement

3.9.1 Tree Structure of SQL Update Statement

SQL Update statement use to change exiting value of a data row in data table. Update statement uses column name as mandatory clause and where clause as optional clause. The level of use column name, column value and sub-query has shown in bellow figure using tree structure.

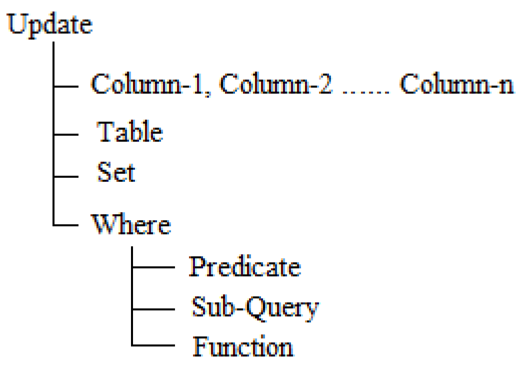

Fig. 3.10: Tree Structure of SQL Update statement

3.9.2 Complexity Value of Database Clauses with Update Statement

Table 3.11: Complexity Value of Different Clauses for Update Statement

Name of the SQL Clause	Complexity Value
Update	U_v
Column	C_v
Set	S_v
Table	t_w
Where	W_v
Predicate	P_v

3.9.3 Complexity Value of SQL Update Statement (C_P)

Complexity value of Update statement depends on the number of mention column and condition. To calculate complexity value for Update statement, we have used the following formula:

$$C_P = U_v + S_v + C_v \times \log_2(1+n) + w_v + P_v + SQV$$

where,

U_v	is the complexity value of UPDATE command
C_v	is the complexity value of column
n	is the number of used columns
w_v	is the complexity value of where clause
P_v	is the complexity value of predicate
SQV	is the complexity of Sub-Query value

SQL Select statement can use as a sub-query. To calculate complexity value of sub-query, we have the formula that use to calculate complexity value of Select statement. More than one column can use with update statement which does not mean to increase complexity value. To limit the complexity value of Update statement, we have used logarithm function with the number of used column.

3.10 Top-down Analysis of SQL Delete Statement

Delete statement uses to remove all or particular rows from data table. Without conditional where clause, it removes all rows from data table. Conditional clause can use Sub-Query to match criteria. SQL select statement uses as sub-query. We have already described details about SQL Select statement.

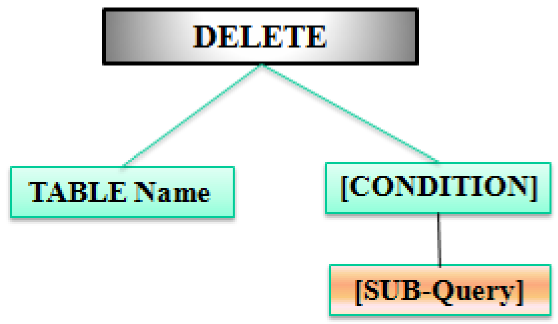

Fig. 3.11: Top-down Analysis of SQL Delete Statement

3.10.1 Tree Structure of SQL Delete Statement

Delete statement uses table name as main clause and where clause as optional. The level of use table name, where clause, predicate, function and sub-query has shown in bellow figure using tree structure.

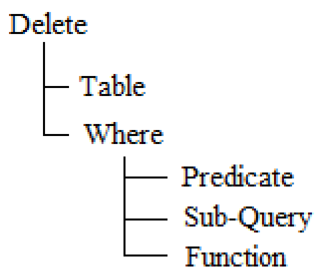

Fig. 3.12: Tree Structure of SQL Delete Statement

3.10.2 Complexity Value of Database Clauses with Delete Statement

Table 3.12: Complexity Value of Different Clauses for Delete Statement

Name of the SQL Clause	Complexity Value
Delete	D_v
Table	t_w
Where	w_v
Predicate	P_v
Function	f_v

3.10.3 Complexity Value of SQL Delete Statement (C_P)

Complexity value of Delete statement depends on the number of parameters with optional where clause. To calculate complexity value of Delete statement, we have used the following formula:

$C_P = D_v + t_w + w_v + P_v + F_v + SQV$

where,

D_v	is the complexity value of DELETE command
t_w	is the complexity value of table
w_v	is the complexity value of where clause
P_v	is the complexity value of predicate
F_v	is the complexity value of function
SQV	is the complexity of Sub-Query value

SQL Select statement can use as a sub-query with DELETE operation. To calculate complexity value of sub-query, we have the formula that use to calculate complexity value of Select statement. More than one column can use with update statement which does not mean to increase complexity value. To limit the complexity value of Update statement, we have used logarithm function with the number of used column.

3.11 Complexity Model on Stored Procedure

Stored procedures are set of Structured Query Language (SQL) statements that perform particular task. The general format for stored procedure is –

```
CREATE PROCEDURE <Procedure_Name>
      -- Add the parameters for the stored procedure here
AS
BEGIN
    -- Insert SQL statements for procedure here
END
```

Stored procedure uses insert, delete, update or select statements with input and output type parameters. Complexity of stored procedure depends on the numbers of use input, output parameters and SQL statements. To calculate complexity value of stored procedure, we have to find out the complexity value of use input, output parameter and SQL statements. We have already described how to calculate complexity value of SQL statements.

Chapter 4
Result and Evaluation

The objective of this chapter is to verify the accuracy and effectiveness of our proposed Complexity Model. The experimental evaluation has been performed using question bank of database lab for undergraduate student. The experimental result has been compared with existing manual system question distribution.

4.1 Experimental Environment

Our proposed Complexity Model has been implemented on a machine (treated as server) with 2.10GHz Intel Core 2 Duo processor and 4GB of RAM, running on Microsoft Windows Server 2008 with Apache server. We have developed a client-server online system for database practical class. The system has been developed in open source environment. We have used PHP for sever side processing and HTML for client side. System administrator has submitted question bank and other related data by using online administrators interface in web browser. For storing and retrieving data we have used Oracle database.

4.2 Complexity Value for Individual Database Item

We have collected the complexity value of individual database clause based on the item type and use position in SQL statement from database specialists. Then we have applied different values for individual database item and find out the most approving value to calculate complexity of a problem. We have calculated SQL problem complexity using the most approving value. In the following table, we have shown details about the complexity value of individual database item.

Table 4.1: Complexity Value for Individual Database Item

SQL Item Details	Level in Use	Complexity Value By DB Expert One	Complexity Value By DB Expert Two	Complexity Value By DB Expert Three	Use Complexity Value
CREATE		3	4	3	3.3
INSERT		3	3	3	3
UPDATE		3	3	3	3
DELETE		3	3	1	2.3
SELECT		3	3	2	2.6
WHERE	SELECT	3	3	4	3.3
GROUP BY		3	4	3	3.3

ORDER BY		3	2	5	3.3
HAVING		3	4	3	3.3
TABLE	FORM	3	3	2	3.6
COLUMN	SELECT	2	2	2	2
	GROUP BY	3	4	3	3.5
	ORDER BY	3	2	2	2
	INSERT	3	3	2	2
	UPDATE	3	3	4	3.3
	CREATE	3	3	3	3
LIKE	WHERE	5	4	3	4
DISTINCT	SELECT	3	3	2	3.6
IN	WHERE	3	3	2	3.6
ROWNUM	SELECT	2	2	2	2
	WHERE	3	3	2	3.6
EXISTS	WHERE	3	4	5	4
BETWEEN		2	2	2	2
NULL		1	1	1	1.3
AND		1	1	2	1.3
OR		1	1	2	1.3
NOT		1	1	2	1.3
DESC	ORDER BY	1	1	2	1.3
ASC		1	1	2	1.3
ON	FROM	2	2	4	2.6
AS	SELECT	1	2	2	1.6
	FROM	1	2	2	1.6
JOIN	FROM	3	3	4	3.3
>=	WHERE	2	2	2	2
<=		2	2	2	2
<>		2	2	2	2
=		2	2	2	2
SUM	SELECT	3	3	2	2.5
	WHERE	3	3	2	2.5
	HAVING	3	3	3	2.5
AVG	SELECT	3	3	2	2.5
	WHERE	3	3	2	2.5
	HAVING	3	3	3	2.5
COUNT	SELECT	3	3	2	2.5
	WHERE	3	3	2	2.5
	HAVING	3	3	3	2.5
MIN	SELECT	2	2	2	2.5
	WHERE	2	2	2	2.5
	HAVING	2	2	3	2.5
MAX	SELECT	2	2	2	2.5
	WHERE	2	2	2	2.5
	HAVING	2	2	3	2.5
	SELECT	2	2	2	2.5

LENGTH	WHERE	2	2	2	2.5
	HAVING	2	2	3	2.5
EXTRACT	SELECT	4	4	4	3.5
	WHERE	4	4	4	3.5
	HAVING	4	4	5	3.5
CONCAT	SELECT	2	3	4	2.5
PRIMARY KEY	CREATE	2	3	3	2.6
DEFAULT		2	2	2	2
UNIQUE		2	2	2	2
CONSTRAINT		3	3	3	3
CHECK		2	3	3	2.6
FOREIGN KEY		2	3	3	2.6
REFERENCES		2	2	3	2.3
CASCADE		3	3	4	3.3
PARTITION		4	4	5	4.3
NOT NULL		1	1	2	1.3
SET	UPDATE	2	2	3	2.3

4.3 Evaluation Methodology

The equivalence of problems means that the complexity to solve the problem is within a specified boundary. Those problems fall into a specified boundary are all similar problems and any of the problem can be assigned to any student and seem to have equal judgment.

Table 4.2: Complexity Level According to Complexity Value

Assignment No	Level No	Complexity Value	Boundary Value ($C_P \pm €$)
1	1	$9 \leq C_P \leq 11$, $C_P = 10$	10 ± 1
	2	$12 \leq C_P \leq 14$, $C_P = 13$	13 ± 1
	3	$15 \leq C_P \leq 17$, $C_P = 16$	16 ± 1
	4	$18 \leq C_P \leq 20$, $C_P = 19$	19 ± 1
	5	$21 \leq C_P$, $C_P = 22$	[21, 22]
2	1	$23 \leq C_P \leq 27$, $C_P = 25$	25 ± 2
	2	$28 \leq C_P \leq 32$, $C_P = 30$	30 ± 2
	3	$33 \leq C_P \leq 37$, $C_P = 35$	35 ± 2
	4	$38 \leq C_P \leq 42$, $C_P = 40$	40 ± 2
	5	$43 \leq C_P \leq 47$, $C_P = 45$	45 ± 2
3	1	$48 \leq C_P \leq 52$, $C_P = 50$	50 ± 2
	2	$53 \leq C_P \leq 57$, $C_P = 55$	55 ± 2
	3	$58 \leq C_P \leq 62$, $C_P = 60$	60 ± 2

	4	$63 \le C_P \le 67, C_P = 65$	65 ± 2
	5	$68 \le C_P \le 72, C_P = 70$	70 ± 2
4	1	$73 \le C_P \le 77, C_P = 75$	75 ± 2
	2	$78 \le C_P \le 82, C_P = 80$	80 ± 2
	3	$83 \le C_P \le 87, C_P = 85$	85 ± 2
	4	$88 \le C_P \le 92, C_P = 90$	90 ± 2
	5	$93 \le C_P \le 100, C_P = 95$	95 ± 2

4.4 Complexity Value of SQL Select Statement

The complexity value of SQL Select statement depends on the number of used clauses, tables, columns, functions, predicates and expression. To observe the effect of increase individual clause, we have analyzed one by one all of those clauses one by one. When we have increased table number then we try to keep fix value for others clause like function, column and other.

4.4.1 Complexity Value of Table in Select Statement

Complexity value of SQL problem depends on the number of used tables to solve the problem. Complex problem requires more tables to find out desired data from database. We have assigned complexity value of table (t_w) to calculate the complexity of a SQL problem. We have applied different t_w value to get more approving result. We have got the best result, when we used $t_w = 3.6$.

Table 4.3: Comparison of Different Complexity Values of Table

Complexity value of Table (t_w)	**% of Similarity compare with SQL-LES**
3.2	86.6
3.3	90.0
3.6	91.6
3.7	90.0
3.8	83.3

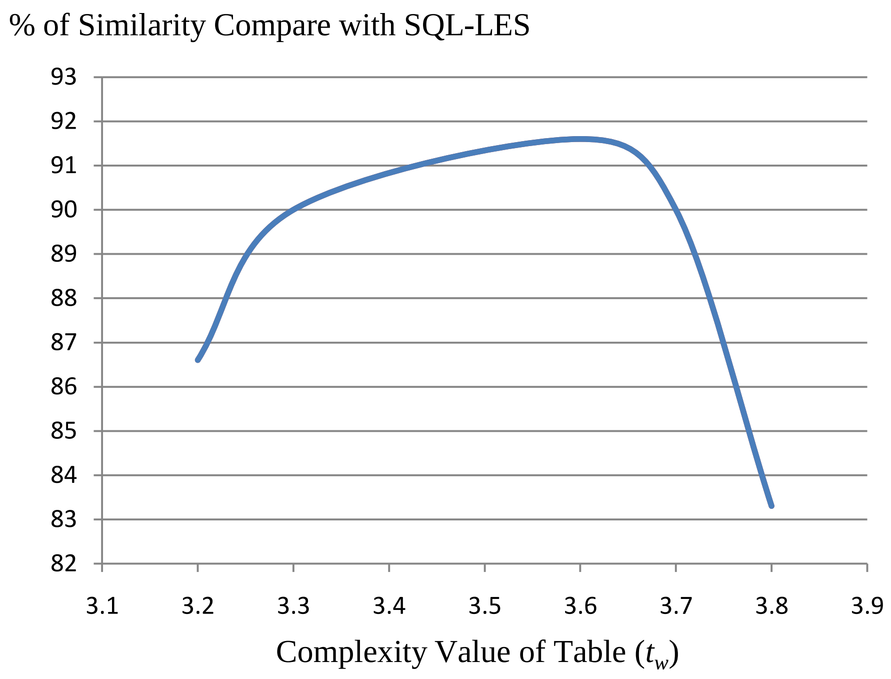

Fig. 4.1: Comparison of Different Complexity Values of Table

We have used the following formula to calculate complexity value of used table:

$$T_v = t_w \times \log_2(1 + n)$$

Table 4.4: SQL-Select Query by Varying Number of Tables

Number of Table	**SQL Statement**
1	`Select  ProductNumber, Name, ListPrice, Size from Product`

2	`SELECT  ProductNumber, P.Name, PM.ModelName, ListPrice FROM Product P`
	`INNER JOIN ProductModel PM ON P.ProductModelID = PM.ProductModelID`

3	`SELECT  ProductNumber, PM.ModelName, OrderQty, CustomerID FROM Product] P`
	`INNER JOIN ProductModel PM ON P.ProductModelID = PM.ProductModelID`
	`INNER JOIN SalesOrderDetail] SOD ON P.ProductID = SOD.ProductID`

5	`SELECT  ProductNumber, PM.ModelName, OrderQty, CD.LastName FROM Product P`
	`INNER JOIN ProductModel PM ON P.ProductModelID = PM.ProductModelID`
	`INNER JOIN SalesOrdDetail SD ON P.ProductID = SD.ProductID`
	`INNER JOIN SalesOrdHeader SOH ON SD.SalesOrdID = SOH.SalesOrdID`
	`INNER JOIN Customer CD ON SOH.CustomerID = CD.CustomerID`

7	`SELECT  ProductNumber, PM.ModelName, OrderQty, ad.PostalCode FROM Product P`
	`INNER JOIN ProductModel PM ON P.ProductModelID = PM.ProductModelID`
	`INNER JOIN SalesOrdDetail SD ON P.ProductID = SD.ProductID`
	`INNER JOIN SalesOrdHeader SH ON SD.SalesOrderID = SH.SalesOrderID`
	`INNER JOIN Customer] CD ON SOH.CustomerID = CD.CustomerID`
	`INNER JOIN CustomerAddress CA ON SOH.CustomerID = CA.CustomerID`
	`INNER JOIN Address Ad ON CA.AddressID= Ad.AddressID`

Complexity value for problem-1:

Table value: $T_v = 3.6 \times \log_2(1+1) = 3.6$

Column value: $C_v = 2 \times \log_2(1+4) \approx 5$

Complexity Value $C_P = 3.6 + 5 = 8.6$

Complexity value for problem-2:

Table value: $T_v = 3.6 \times \log_2(1+2) \approx 5.7$

Column value: $C_v = 2 \times \log_2(1+4) \approx 5$

Predicate value: $Pv = 2$

Complexity Value: $C_P = 5.7 + 5 + 2 = 12.7$

Complexity value for problem-3:

Table value: $T_v = 3.6 \times \log_2(1+7) \approx 10.8$

Column value: $C_v = 2 \times \log_2(1+4) \approx 5$

Predicate value: $Pv = 2$

Complexity Value: $C_P = 10.8 + 5 + 2 = 17.8$

Table 4.5: Complexity Value of SQL Problems by Varying Number of Tables

Number of Table	Number of Column	Number of Function	Number of predicate	Number of Expression	Complexity Value
1	4	0	0	0	8.6
2	4	0	2	0	12.7
3	4	0	4	0	14.8
5	4	0	8	0	16.35
7	4	0	12	0	17.8

The complexity value of SQL problem by varying the number of used table has shown on above table. In the above example the number of table has increase, but the syntax of table joining is same. It does not means that the complexity value of SQL problem will proportionally increase with the number of used table. To curb the complexity value of table, we have used logarithm function which has helped to calculate accurate complexity value.

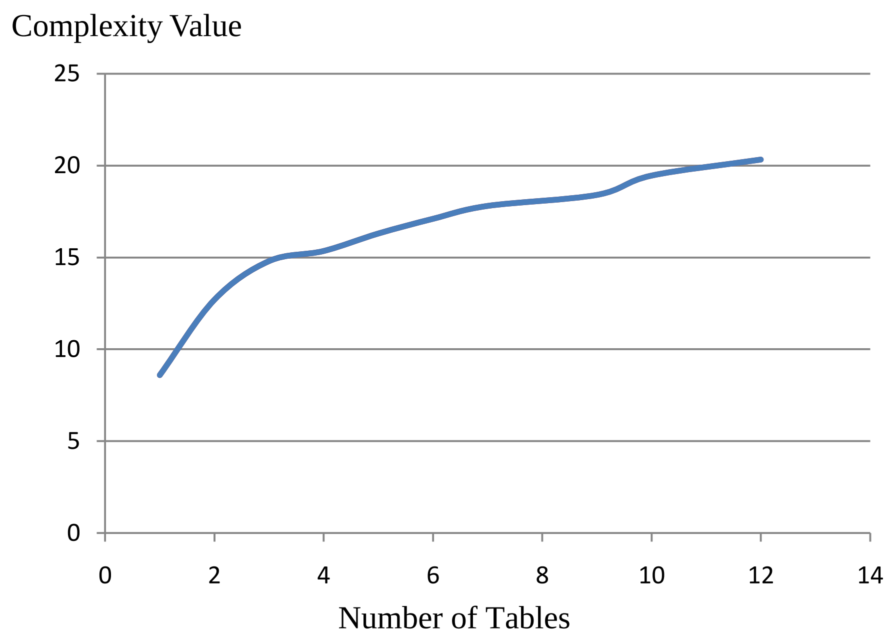

Fig. 4.2: Complexity Value of SQL Problems by Varying Number of Tables

The complexity value of SQL problem has logarithmic increase with the number of used table. For huge number of used table, complexity value has increased minimally.

4.4.2 Complexity Value of Columns in Select Statement

Used column in SQL query plays vital role to calculate complexity value of SQL Select statement. Colum can use with SELECT, GROUP BY or ORDER BY clause. Complexity value of used column depends where it is use. Order of column on GROUP BY or ORDER By clause has different meaning. Complexity value of column with SELECT clause has different value than GORUP BY or ORDER BY clause.

Table 4.6: Comparison of Different Complexity Values of Column with Select Clause

Level in use	**Complexity value of Column (C_{vS})**	**% of Similarity compare with SQL-LES**
Select	1	83.3
	1.5	85
	2	91.6
	2.3	81.6
	2.5	63.3

Different complexity value of column and their effectiveness has shown in the table 4.6. We have collected complexity value of column from different database specialist then we have applied different complexity value base on collected value. We have got maximum approving at C_{vS} =2.0. Using the maximum approving value, it can calculate 91% of similar result as SQL-LES.

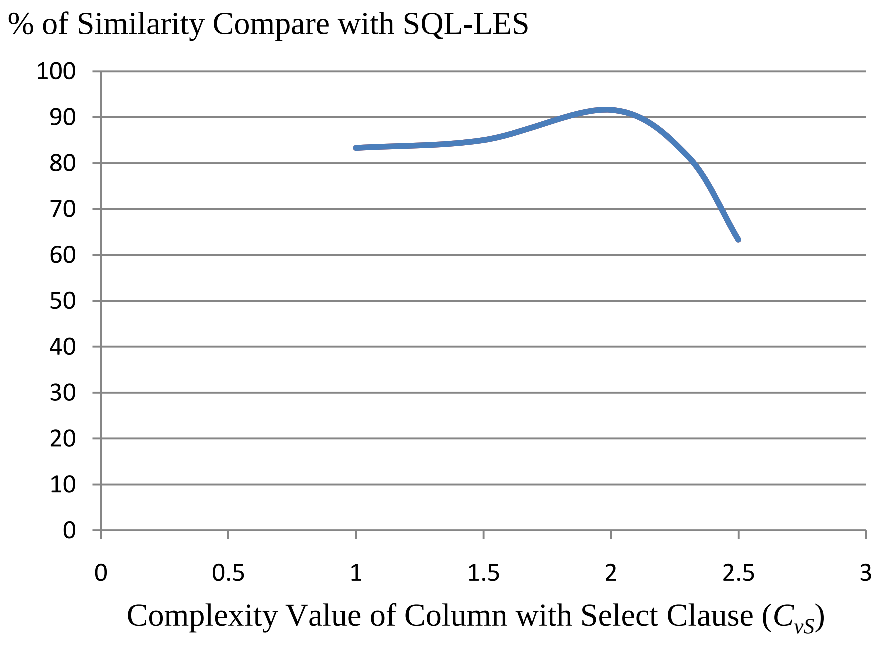

Fig. 4.3: Comparison of Different Complexity Values of Column with Select Clause

Column has different meanings base on their use position. Group By clause can generate different result base the use column and their order of use. We have analyzed the complexity value of column using value to observe the best approving value. It can calculate 91% of similar result compare to existing SQL-LES at C_{vG} =3.5.

Table 4.7: Comparison of Different Complexity Values of Column with Group By Clause

Level in use	**Complexity value of Column (C_{vG})**	**% of Similarity compare with SQL-LES**
Group By	2.5	90
	3	91.6
	3.3	91.6
	3.5	91.6
	4	90

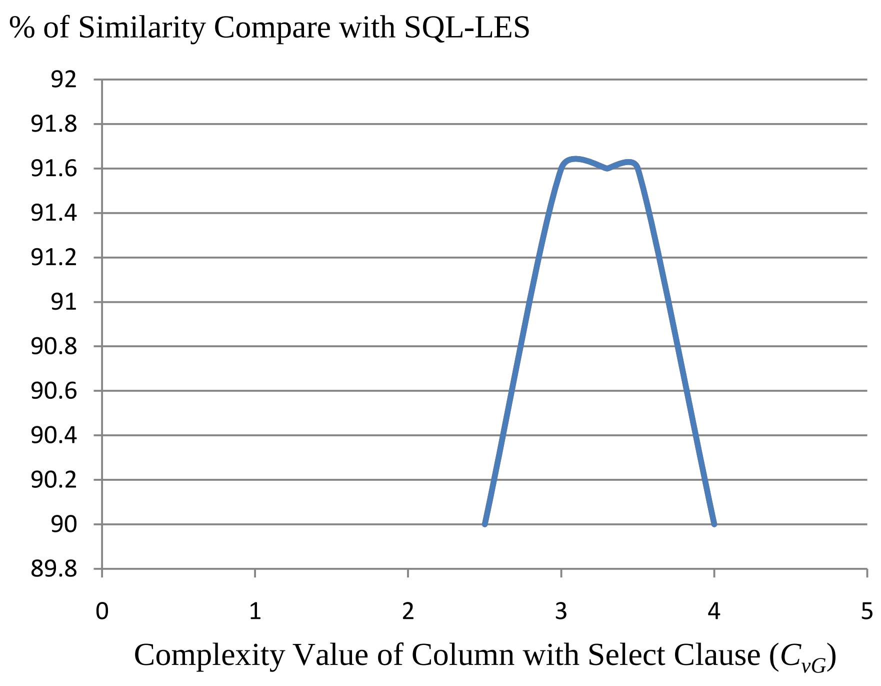

Fig. 4.4: Comparison of Different Complexity Values of Column with Group By Clause

Column has different meanings base on their use position. Order By clause uses column name to make the retrieve data more meaningful. We have analyzed the complexity value of column with different values. It can calculate 91% of similar result compare to existing SQL-LES at C_{vO} =2.0.

Table 4.8: Comparison of Different Complexity Values of Column with Order By Clause

Level in use	**Complexity value of Column (C_{vO})**	**% of Similarity compare with SQL-LES**
Order By	1.5	88.3
	2.0	91.6
	2.5	90
	3.0	86.6
	3.5	80

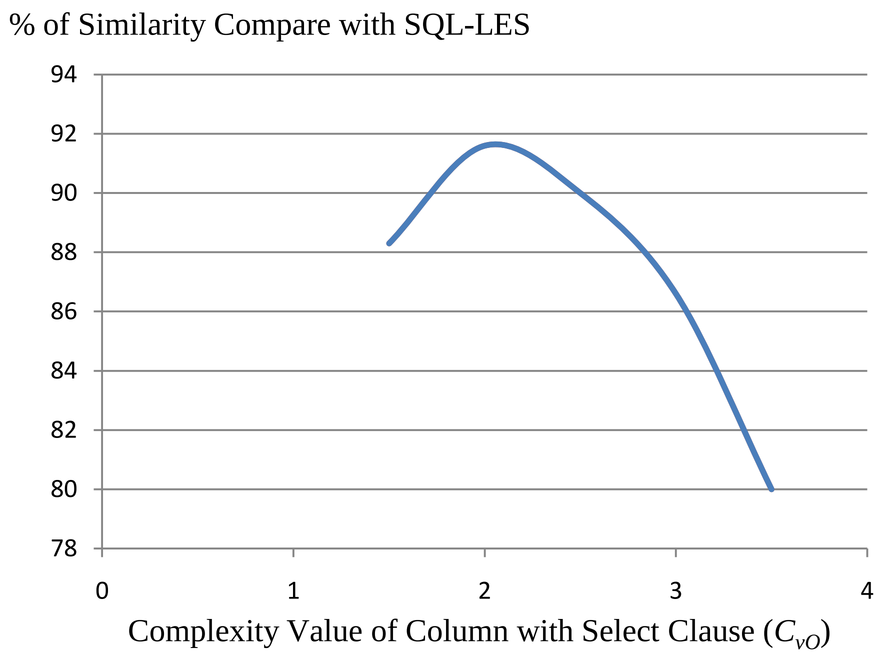

Fig. 4.5: Comparison of Different Complexity Values of Column with Order By Clause

Following formula has been used to calculate complexity valued of used column in SQL SELECT statement.

$$C_v = c_{vS} \times \log_2(1+m) + c_{vG} \times \log_2(1+n) + c_{vO} \times \log_2(1+p)$$

Table 4.9: SQL Select Query by Varying Number of Columns

Number of Table	SQL Statement
$C_S = 3$ $C_O = 1$	Select **ProductNumber,** **Name,** **ListPrice** from Product where Color = 'Silver' **ORDER BY** ProductNumber
$C_S = 5$ $C_O = 2$	**Select** **ProductNumber,** **Name,** **Class,** **Size,** **ListPrice from Product where Color = 'Silver'** **ORDER BY Class, Size**
	SELECT **ProductNumber,**

$C_S = 6$ $C_O = 3$	**Name,** **Class,** **Size,** **Style,** **ListPrice from Product where Color = 'Silver'** **ORDEER BY Class, Size, Style**
$C_S = 7$ $C_O = 3$	**SELECT** **ProductNumber,** **Name,** **Class,** **Size,** **Style,** **Weight,** **ListPrice from Product where Color = 'Silver'** **ORDEER BY Class, Size, Style**

Complexity value for problem-1:

Table value: $T_v = 10 \times \log_2(1+1) = 10$

Column value: $C_v = 2 \times \log_2(1+3) + 3 \times \log_2(1+1) = 7$

Predicate value: $Pv = 1$

Complexity Value: $C_P = 10 + 7 + 1 = 18$

Complexity value for problem-2:

Table value: $T_v = 10 \times \log_2(1+1) = 10$

Column value: $C_v = 2 \times \log_2(1+3) + 3 \times \log_2(1+1) = 10$

Predicate value : $Pv = 1$

Complexity Value: $C_P = 10 + 10 + 1 = 21$

Complexity value for problem-3:

Table value: $T_v = 10 \times \log_2(1+1) = 10$

Column value: $C_v = 2 \times \log_2(1+6) + 3 \times \log_2(1+3) \approx 12$

Predicate value : $Pv = 2$

Complexity Value: $C_P = 10 + 12 + 2 = 24$

Complexity value for problem-4:

Table value: $T_v = 10 \times \log_2(1+1) = 10$

Column value: $C_v = 2 \times \log_2(1+3) + 3 \times \log_2(1+1) = 12$

Predicate value: $P_v = 1$

Complexity value: $C_P = 10 + 7 + 1 = 23$

Table 4.10: Complexity Value of SQL Problems by Varying Number of Columns

Number of Table	Number of Column	Number of Function	Number of predicate	Number of Expression	Complexity Value
1	4	0	1	0	18
1	7	0	1	0	21
1	9	0	1	0	23
1	10	0	1	0	23

Table 4.6 has shown the complexity value of SQL problem by varying the number of used column with different SQL clause. Column usages with different clause have different meaning. SELECT clause can use column with any order which does not effect on SQL query result, but column ordering with GROUP BY or ORDER BY clause has different meaning. For this reason, we assume different complexity value of column with different clause.

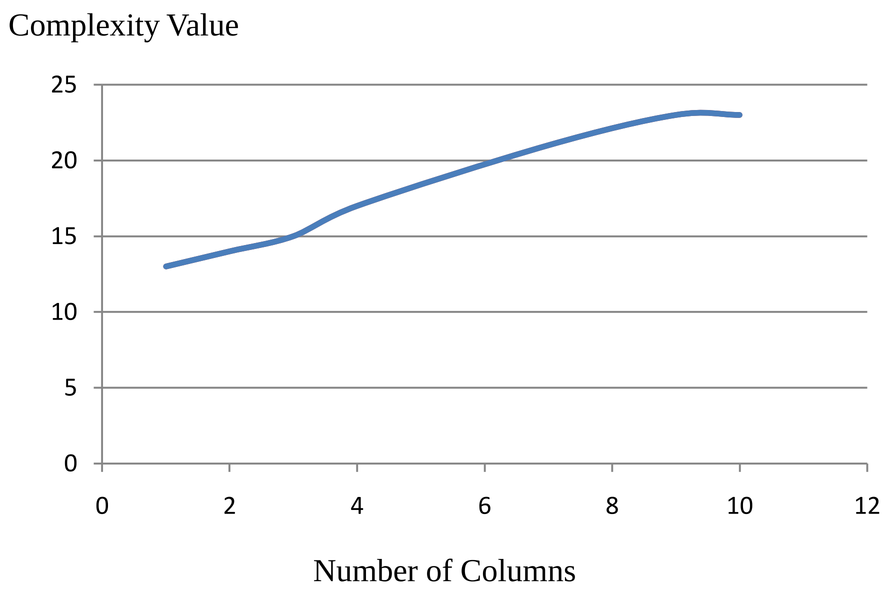

Fig. 4.6: Complexity Value of SQL Problems by Varying Number of Columns

The complexity value of SQL problem has increase logarithmically with the number of used column. For huge number of used table, complexity value has increased minimally

4.4.3 Complexity Value of Functions in Select Statement

Lots of functions use with SQL query to calculate, format or convert data. The main purpose of function in query is to represent stored data as meaningful information. Lots of functions are similar type i.e. number of used parameter and working mechanism is same. So the Complexity value of SQL problem depends on how many similar types of functions has been used to solve the problem. We have analyzed complexity value of functions by applying different parameter value for those functions.

Table 4.11: Comparison of Different Complexity Values for Type-1 Functions

Function Type	**Complexity value of Function (f_w)**	**% of Similarity compare with SQL-LES**
Sum, Avg, Count and Concat	1.5	88.3
	2.0	90
	2.5	91.6
	3	86.6
	3.5	81.6

Different types of function use with SQL select statement. We have grouped those function in different types base on the return type and the number of function parameters. To find out more approving complexity value of function, we have analysed different parameter value rang from 1.5 to 3.5 for type-1 functions. We have found that f_w = 2.5 give the best result for type-1 function. We have shown the analysed result in the table 4.11.

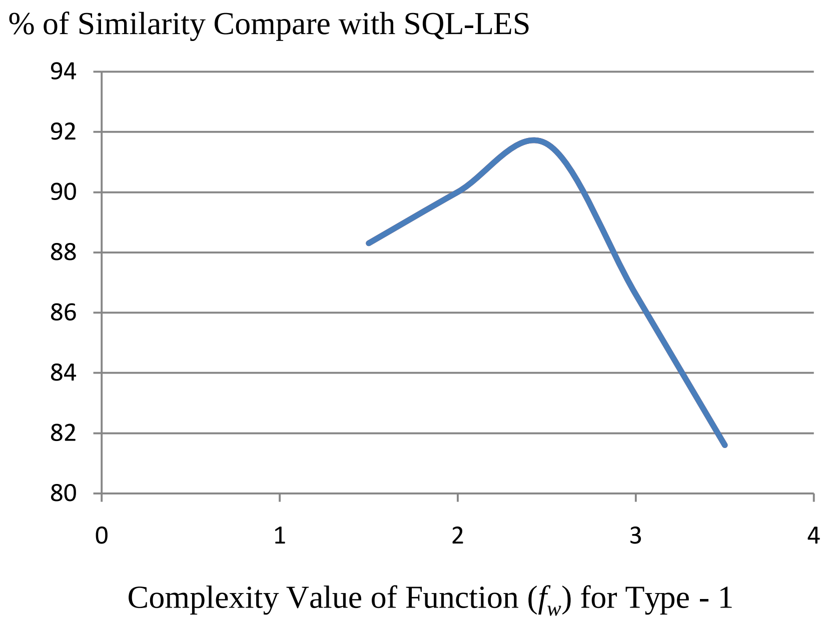

Fig. 4.7: Comparison of Different Complexity Values for Type-3 Functions

We have shown the analysed result of type-2 function in the table 4.12. Different functions use with SQL select statement for this type. To find out more approving complexity value of those functions, we have analysed different parameter value rang from 1.5 to 3.5 for type-2 functions. We have found that f_w = 2.5 give the best result for type-2 function.

Table 4.12: Comparison Result using Different Complexity Values for Type-2 Functions

Function Type	**Complexity value of Function (f_w)**	**% of Similarity compare with SQL-LES**
Min, Max, Length and Floor	1.5	86.6
	2.0	90
	2.5	91.6
	3	90
	3.5	86.6

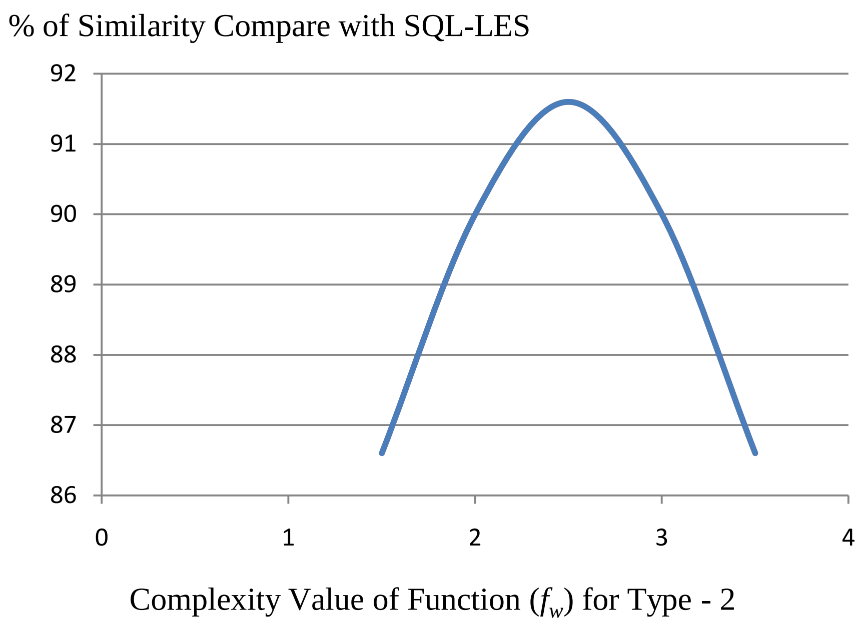

Fig. 4.8: Comparison of Different Complexity Values for Type-3 Functions

We have shown the analysed result of type-3 function in the table 4.13. Different functions use with SQL select statement for this type. To find out more approving complexity value of those functions, we have analysed different parameter value rang from 2.0 to 4.5 for type-3 functions. We have found that f_w = 3.5 give the best result for this type of functions.

Table 4.13: Comparison Result using Different Complexity Values for Type-3 Functions

Function Type	**Complexity value of Function (f_w)**	**% of Similarity compare with SQL-LES**
Extract, Substr and Month_between	2.0	90
	2.5	91.6
	3.5	91.6
	4.0	91.6
	4.5	90

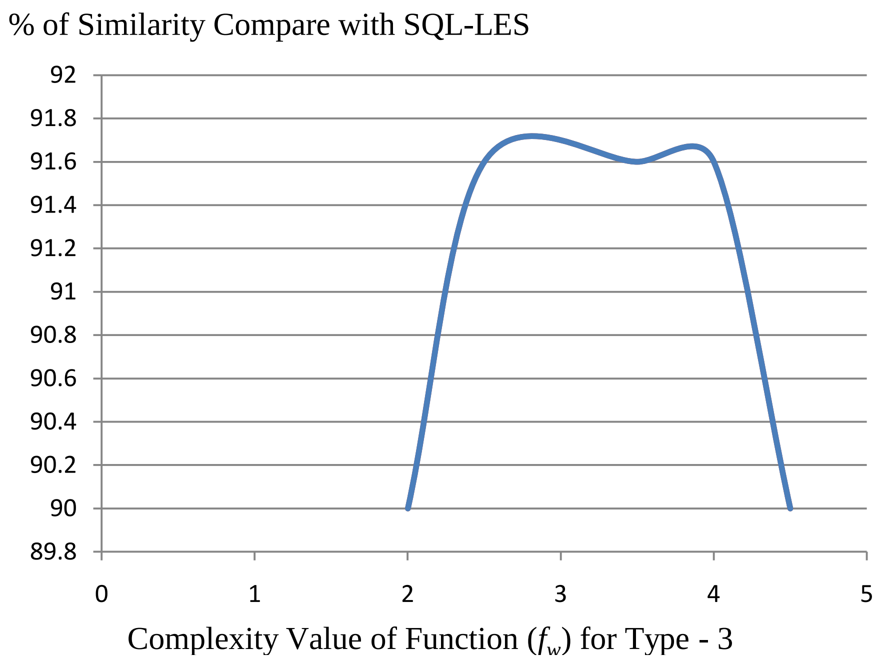

Fig. 4.9: Comparison of Different Complexity Values for Type-3 Functions

We have used the following formula to calculate complexity value of function:

$$F_v = \sum_{t=0}^{m} f_{wt} \times \log_2(1+n)$$

Table 4.14: SQL Select Query by Varying Number of Functions

Number of Function	SQL Statement

1	`SELECT ProductID,`
	 `COUNT(SalesOrderID) as [Total Order]`
	 `FROM SalesOrderDetail`
	 `GROUP BY ProductID`
	 `ORDER BY ProductID`

2	`SELECT ProductID,`
	 `COUNT(SalesOrderID) as [Total Order],`
	 `SUM(OrderQty) as [Order Quantity]`
	 `FROM SalesOrderDetail`
	 `GROUP BY ProductID`
	 `ORDER BY ProductID`

4	`SELECT ProductID`
	 `,COUNT(SalesOrderID) as [Total Order]`
	 `,SUM(OrderQty)[Order Quantity]`
	 `,AVG(UnitPrice)[Avg Unit Price]`
	 `,SUM(OrderQty*UnitPrice) [Total Price]`
	 `FROM SalesOrderDetail`

	`GROUP BY ProductID`
	`ORDER BY [Total Order] desc`

Complexity value for problem-1:

Table value: T_v = $10 \times \log_2(1+1) = 10$

Column value: C_v = $2 \times \log_2(1+2) + 3 \times \log_2(1+2) \approx 8$

Function value: fv = $3 \times \log_2(1+1) = 3$

Predicate value : Pv = 1

Expression value: Ev = 1

Complexity Value C_P = 10 + 8 + 3 + 1 + 1 = 23

Complexity value for problem-2:

Table value: T_v = $10 \times \log_2(1+1) = 10$

Column value: C_v = $2 \times \log_2(1+3) + 3 \times \log_2(1+2) \approx 9$

Function value: fv = $3 \times \log_2(1+2) \approx 5$

Predicate value : Pv = 1

Expression value: Ev = $1 \times \log_2(1+2) \approx 2$

Complexity Value C_P = 10 + 9 + 5 + 1 + 2 = 27

Complexity value for problem-3:

Table value: T_v = $10 \times \log_2(1+1) = 10$

Column value: C_v = $2 \times \log_2(1+5) + 3 \times \log_2(1+2) \approx 10$

Function value: fv = $2 \times \log_2(1+2) + 3 \times \log_2(1+2) \approx 8$

Predicate value : Pv = 1

Expression value: Ev = $1 \times \log_2(1+4) + 2 \times \log_2(1+1) \approx 4$

Complexity Value C_P = 10 + 10 + 8 + 1 + 4 = 33

Table 4.15: Complexity Value of SQL Problems by Varying Number of Functions

Number of Table	Number of Column	Number of Function	Number of predicate	Number of Expression	Complexity Value
1	4	1	1	1	23

1	5	2	2	2	27
1	7	4	2	5	33

Table 4.8 has shown the complexity value of SQL problem by varying the number of used similar type function. In the above example, we have calculated how many similar type functions used to process the query. Most of the time function uses column, predicate and other items. So the complexity value of other items can be increase with the number of function. Complexity value of Column and Expression has increased in the above example.

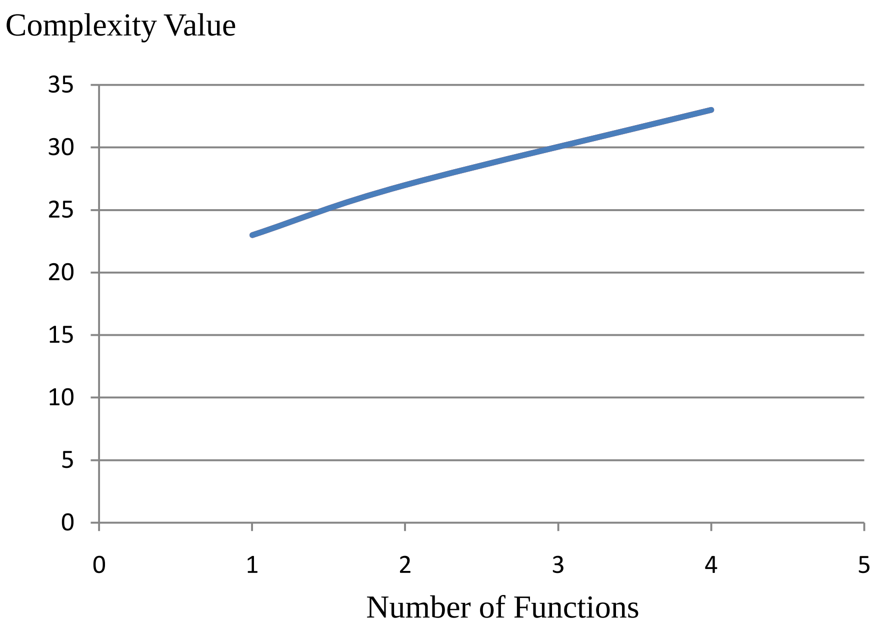

Fig. 4.10: Complexity Value of SQL Problems by Varying Number of Functions

The complexity value of SQL problem has logarithmic increase with the number of used table. For huge number of used table, complexity value has increased minimally

4.4.4 Complexity Value of Predicates in Select Statement

Rang of simple to complex predicate are use with SQL query to retrieve data. Similar type of predicate can use anywhere in SQL query. Predicate complexity depends on the type of predicate. We have analyzed the complexity value of predicate by applying different parameter value to define the more approving value.

Table 4.16: Comparison of Different Complexity Values for Comparison Type Predicates

Predicate Type	**Complexity value of**	**% of Similarity compare**

	Column (p_w)	**with SQL-LES**
Comparison Predicate	1.0	83.3
	1.5	86.6
	2	91.6
	2.5	86.6
	3	83.3

We have shown the complexity value of comparison type predicates in the tbale 4.16. We have analysed complexity value by applying different parameter value range from 1.0 to 3.0. We have got the best result at p_w = 2.

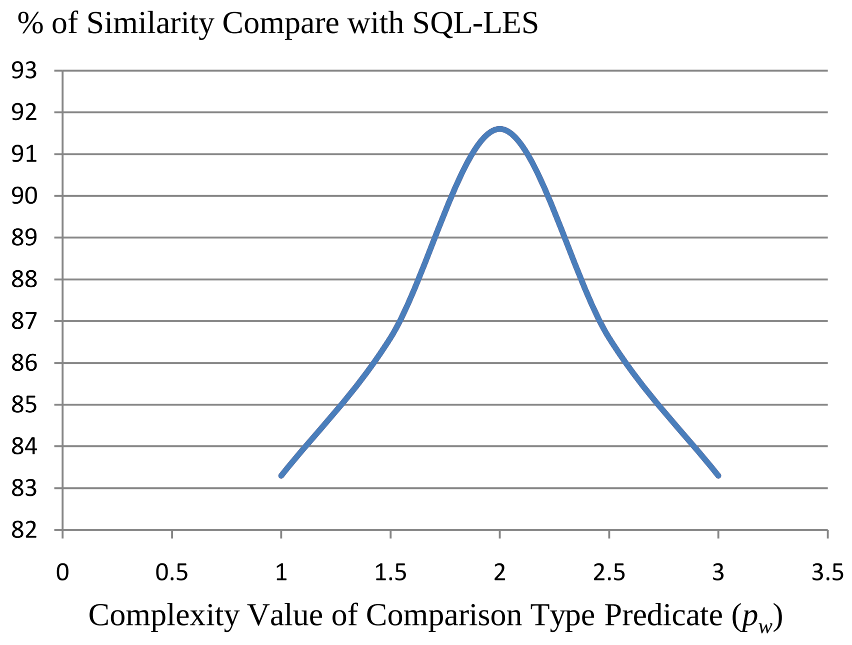

Fig. 4.11: Comparison of Different Complexity Values for Comparison Type Predicates

We have shown the complexity value of logical type predicates in the tbale 4.17. We have analysed complexity value by applying different parameter value range from 1.0 to 1.5. We have got the best result at p_w = 1.3.

Table 4.17: Comparison of Different Complexity Values for Logical Type Predicates

Predicate Type	**Complexity value of Predicate (p_w)**	**% of Similarity compare with SQL-LES**
Logical Predicate	1.0	90
	1.2	91.6
	1.3	91.6
	1.5	90

	1.8	86.6

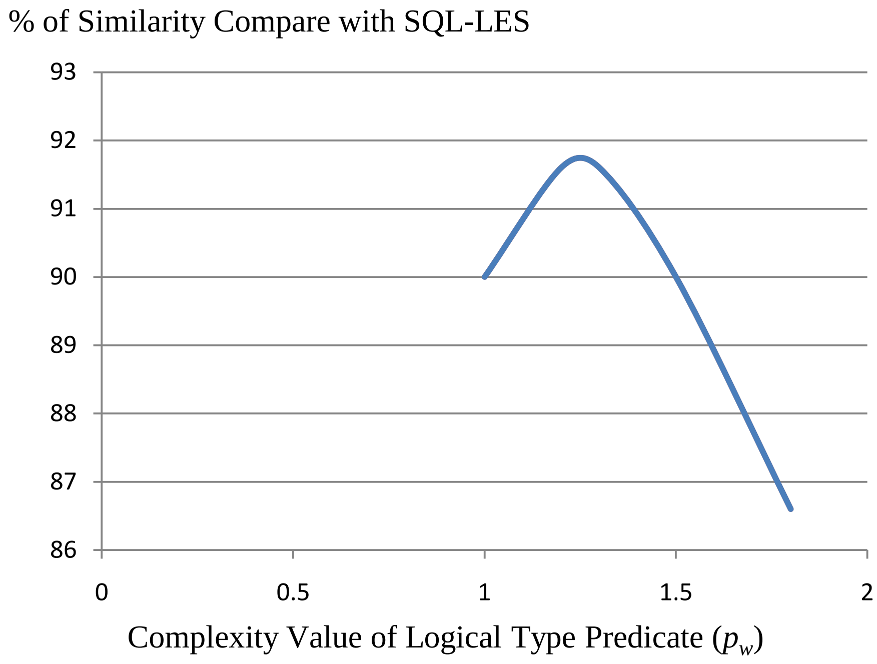

Fig. 4.12: Comparison of Different Complexity Values for Logical Type Predicates

We have used the following formula to calculate complexity value of predicate:

$$P_v = \sum_{t=0}^{n} p_{wt} \times \log_2(1+n)$$

Table 4.18: SQL Select Query by Varying Number of Predicates

Number of Predicate	SQL Statement
1	`SELECT DISTINCT Style, Name, Size FROM Product`
2	`SELECT DISTINCT Style, Name, Size FROM Product WHERE Size in ('S','M','L')`
5	`SELECT DISTINCT Style, Name, Size FROM Product WHERE Size in ('S','M','L') AND Style IS NOT NULL`
7	`SELECT DISTINCT Style, Name, Size FROM Product WHERE Size IN ('S','M','L') AND Style IS NOT NULL AND Name LIKE '%Classic%'`
9	`SELECT DISTINCT Style, Name, Size FROM Product WHERE Size IN ('S','M','L') AND Style IS NOT NULL AND (Name LIKE '%Classic%' OR Name LIKE '%Women%' )`
12	`SELECT DISTINCT Style, Name, Size FROM Product WHERE Size IN ('S','M','L') AND Style IS NOT NULL AND (Name LIKE '%Classic%' OR Name LIKE '%Women%' ) AND SellStartDate BETWEEN '2002-07-01 00:00:00.000' AND '2002-07-31 23:59:59.000'`

Complexity value for problem-1:

Table value: $T_v = 10 \times \log_2(1+1) = 10$

Column value: $C_v = 2 \times \log_2(1+3) \approx 4$

Predicate value: $P_v = 3$

Complexity Value $C_P = 10 + 4 + 3 = 17$

Complexity value for problem-2:

Table value: $T_v = 10 \times \log_2(1+1) = 10$

Column value: $C_v = 2 \times \log_2(1+3) \approx 4$

Predicate value: $P_v = 4$

Complexity Value $C_P = 10 + 4 + 4 = 18$

Complexity value for problem-3:

Table value: $T_v = 10 \times \log_2(1+1) = 10$

Column value: $C_v = 2 \times \log_2(1+3) \approx 4$

Predicate value: $P_v = 7$

Complexity Value $C_P = 10 + 4 + 7 = 21$

Complexity value for problem-4:

Table value: $T_v = 3.6 \times \log_2(1+1) = 3.6$

Column value: $C_v = 2 \times \log_2(1+3) \approx 4$

Predicate value: $P_v = 12$

Complexity Value $C_P = 3.6 + 4 + 12 = 19.9$

Complexity value for problem-5:

Table value: $T_v = 10 \times \log_2(1+1) = 10$

Column value: $C_v = 2 \times \log_2(1+3) \approx 4$

Predicate value: $P_v = 13$

Complexity Value $C_P = 10 + 4 + 7 = 27$

Complexity value for problem-6:

Table value: $T_v = 10 \times \log_2(1+1) = 10$

Column value: $C_v = 2 \times \log_2(1+3) \approx 4$

Predicate value: $P_v = 18$

Complexity Value $C_P = 10 + 4 + 7 = 32$

Table 4.19: Complexity Value of SQL Problem by Varying Number of Predicates

Number of Table	Number of Column	Number of Function	Number of predicate	Number of Expression	Complexity Value
1	3	0	1	0	17
1	3	0	2	0	18
1	3	0	5	0	21
1	3	0	7	0	26
1	3	0	9	0	27
1	3	0	12	0	32

Table 4.10 has shown the complexity value of SQL problem by varying the number of used predicate. In the above example most of the predicate repeated, but the syntax of their usage is same. To calculate complexity value of predicate, we have grouped similar type predicates.

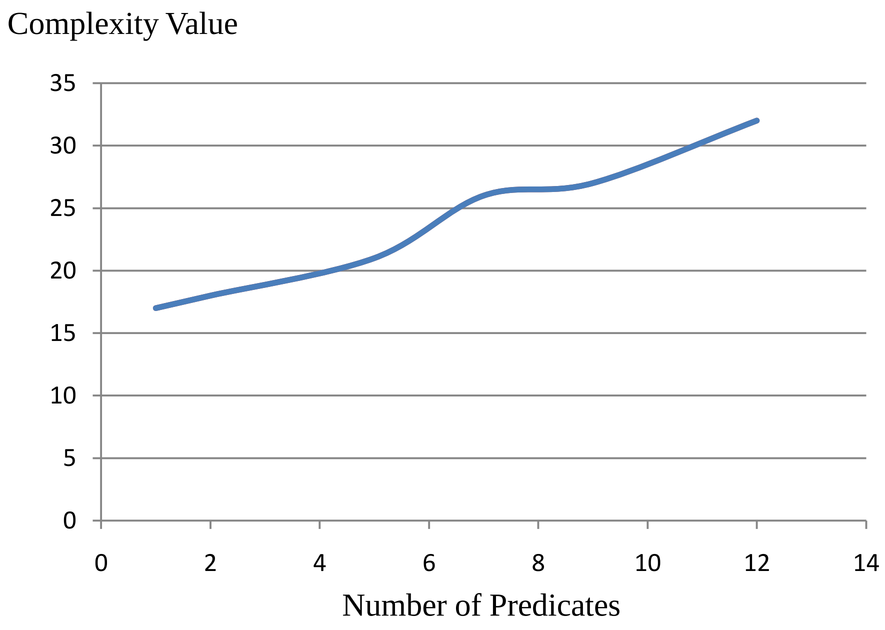

Fig. 4.13: Complexity Value of SQL Problems by Varying Number of Predicates

The complexity value of SQL problem has increased logarithmically increase with the number of used table. For the huge number of used predicate, complexity value has increased minimally.

4.4.5 Complexity Value by Increasing Table, Column, Function, Predicate and Expression for Select Statement

All examples on above sections have shown the complexity value by varying individual item like table, function and others only. In this section we have shown the complexity value by increasing table, column, function, and others.

Table 4.20: SQL Select Query by Increasing Parameters of All Clauses

No.	SQL Statement
1	`Select  ProductNumber, Name, ListPrice from Product where Color = 'Silver'`
2	`SELECT  ProductNumber, P.Name, PM.Name AS [Model Name], ListPrice FROM Product P INNER JOIN ProductModel PM ON P.ProductModelID = PM.ProductModelID WHERE Color = 'Silver' order by PM.Name`
3	`SELECT ProductNumber, P.Name, PM.Name AS [Model Name], ListPrice, OrderQty, SOH.OrderDate FROM Product P INNER JOIN ProductModel PM ON P.ProductModelID = M.ProductModelID INNER JOIN SalesOrderDetail SOD ON P.ProductID = SOD.ProductID INNER JOIN SalesOrderHeader SOH ON SOD.SalesOrderID = SOH.SalesOrderID WHERE Color = 'Silver' and SOH.OrderDate >='2004-06-01' and SOH.OrderDate <='2004-07-01'`
4	`SELECT ProductNumber, P.Name, PM.Name AS [Model Name], ListPrice, OrderQty, ListPrice*OrderQty as [Total Price], SOH.OrderDate FROM Product P INNER JOIN ProductModel PM ON P.ProductModelID = PM.ProductModelID AND PM.Name in ('Front Brakes','LL Mountain Frame','Mountain-500') INNER JOIN SalesOrderDetail SOD ON P.ProductID = SOD.ProductID INNER JOIN SalesOrderHeader SOH ON SOD.SalesOrderID = SOH.SalesOrderID WHERE Color = 'Silver' and SOH.OrderDate >='2004-06-01' and SOH.OrderDate <='2004-07-01' ORDER BY OrderQty DESC`
5	`SELECT ProductNumber, P.Name, PM.Name AS [Model Name], ListPrice, OrderQty, ListPrice*OrderQty as [Total Price], SOH.OrderDate FROM Product P INNER JOIN ProductModel PM ON P.ProductModelID = PM.ProductModelID AND PM.Name in ('Front Brakes','LL Mountain Frame','Mountain-500') INNER JOIN SalesOrderDetail SOD ON P.ProductID = SOD.ProductID AND SOD.OrderQty > (SELECT MIN(OrderQty) from SalesOrderDetail) INNER JOIN SalesOrderHeader SOH ON SOD.SalesOrderID = SOH.SalesOrderID WHERE Color = 'Silver' and SOH.OrderDate >='2004-06-01' and SOH.OrderDate <='2004-07-01' ORDER BY OrderQty, [Total Price] DESC`
6	`SELECT SOH.CustomerID, CSD.FirstName + ', ' + CSD.LastName as [Full Name], Phone, EmailAddress, P.Name, PM.Name AS [Model Name], ListPrice, OrderQty, ListPrice*OrderQty as [Total Price], SOH.OrderDate FROM Product P INNER JOIN ProductModel PM ON P.ProductModelID = PM.ProductModelID AND PM.Name in ('Front Brakes','LL Mountain Frame','Mountain-500') INNER JOIN SalesOrderDetail SOD ON P.ProductID = SOD.ProductID AND SOD.OrderQty > (SELECT MIN(OrderQty) from SalesOrderDetail) INNER JOIN SalesOrderHeader SOH ON SOD.SalesOrderID = SOH.SalesOrderID INNER JOIN Customer CSD ON SOH.CustomerID = CSD.CustomerID WHERE Color = 'Silver' AND P.Name LIKE '%LL Mountain%' AND SOH.OrderDate >='2004-06-01' and SOH.OrderDate <='2004-07-01'`

	`ORDER BY  OrderQty, [Total Price] DESC`

Complexity value for problem-1:

Table value: T_v = $3.6 \times \log_2(1+1) = 3.6$

Column value: C_v = $2 \times \log_2(1+3) \approx 4$

Predicate value: Pv = 2

Complexity Value C_P = 3.6 + 4 + 2 = 9.6

Complexity value for problem-2:

Table value: T_v = $3.6 \times \log_2(1+2) \approx 5.7$

Column value: C_v = $2 \times \log_2(1+4) + 3 \times \log_2(1+1) \approx 8$

Predicate value: Pv = 3+2+1 = 6

Expression value: E_v = 1

Complexity Value C_P = 5.7 + 8 + 6 + 1 = 20.7

Complexity value for problem-3:

Table value: T_v = $3.6 \times \log_2(1+4) \approx 8.35$

Column value: C_v = $2 \times \log_2(1+6) \approx 6$

Predicate value: Pv = 3+2+1+2+2+1 = 11

Expression value: E_v = 1

Complexity Value C_P = 8.35 + 6 + 11 + 1 = 26.35

Complexity value for problem-4:

Table value: T_v = $3.6 \times \log_2(1+4) \approx 8.35$

Column value: C_v = $2 \times \log_2(1+8) + 3 \times \log_2(1+1) \approx 9$

Predicate value: Pv = 3+2+1+2+2+3 + 1+1= 15

Expression value: E_v = $1 \times \log_2(1+2) + 2 \times \log_2(1+1) \approx 4$

Complexity Value C_P = 8.35 + 9 + 15 + 4 = 36.35

Complexity value for problem-5:

Table value: T_v = $3.6 \times \log_2(1+5) \approx 9.3$

Column value: C_v = $2 \times \log_2(1+9) + 3 \times \log_2(1+2) \approx 11$

Function value: $F_v = 2$

Predicate value: $Pv = 3+2+1+2+2+3 + 1+1+2= 17$

Expression value: $E_v = 1 \times \log_2(1+2) + 2 \times \log_2(1+1) \approx 4$

Complexity Value $C_P = 9.3 + 11 + 2 + 17 + 4 = 43.3$

Complexity value for problem-6:

Table value: $T_v = 3.6 \times \log_2(1+6) \approx 10.1$

Column value: $C_v = 2 \times \log_2(1+12) + 3 \times \log_2(1+2) \approx 11$

Function value: $F_v = 2$

Predicate value: $Pv = 3+2+1+2+2+3 + 1+1+2 + 5 = 22$

Expression value: $E_v = 1 \times \log_2(1+2) + 2 \times \log_2(1+1) + 3 \times \log_2(1+1) \approx 7$

Complexity Value $C_P = 10.1 + 11 + 2 + 22 + 7 = 52.1$

Table 4.21: Complexity Value of Select Statements Varying by Parameters

Number of Table	Number of Column	Number of Function	Number of predicate	Number of Expression	Complexity Value
1	3	0	1	0	9.6
2	5	0	4	1	20.7
4	6	0	12	1	26.35
4	9	0	16	3	36.35
5	11	1	17	3	43.3
6	14	1	18	4	52.1

Table 4.12 has shown the complexity value of SQL problem by increasing all clause items like table, column, function and others. In the above example most of the item has repeated, but the syntax of their usage is same. To calculate complexity value of problem, we have grouped similar type items and used logarithm function to curb the value.

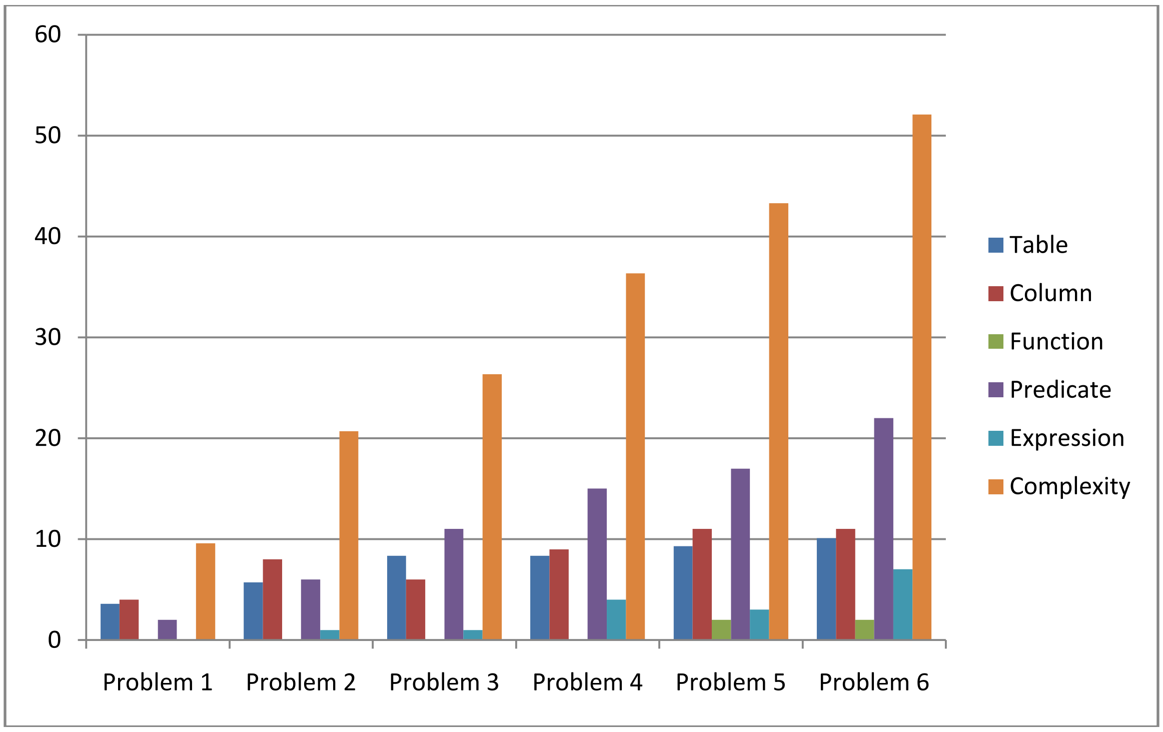

Fig. 4.14: Complexity Value of Select Statements Varying by Parameters

4.5 Complexity Value of SQL Create Statement

The complexity value of SQL Select statement depends on the number of used tables, columns, functions, predicates and expression. To observe the effect of increase individual clause, we have analyzed one by one all of those clauses one by one. When we have increased table number then we try to keep fix value for others clause like function, column and other.

Table 4.22: SQL Create Statements with Different Constraints

No.	SQL Create Statement
1	`create table tablename (col1 number, col2 char(25), col3 number, col4 number(10), col5 char(1));`
2	`create table tablename (col1 number primary key, col2 char(25), col3 number default 0, col4 number(10) unique, col5 char(1));`
3	`create table tablename (col1 number primary key, col2 char(25), col3 number default 0, col4 number(10) unique, col5 char(1),`

	`constraint col5_cst check (col5 in (‘M’,‘F’)));`
4	`create table tablename (col1 number primary key, col2 char(25), col3 number default 0, col4 number(10) unique, col5 char(1), constraint col5_cst check (col5 in (‘M’,‘F’)), constraint col2_fk foreign key (col2) references table2(col1));`
5	`create table tablename (col1 number primary key, col2 char(25), col3 number default 0, col4 number(10) unique, col5 char(1), constraint col5_cst check (col5 in (‘M’,‘F’)), constraint col2_fk foreign key (col2) references table2(col1) on delete cascade);`
6	`create table tablename (col1 number primary key, col2 char(25), col3 number default 0, col4 number(10) unique, col5 char(1), constraint col5_cst check (col5 in (‘M’,‘F’)), constraint col2_fk foreign key (col2) references table2(col1) on delete cascade) partition by reference (col2_fk);`

Complexity value for problem-1:

Create Table: $CT_v = 3$

Column value: $C_v = 3 \times \log_2(1+5) \approx 8$

Complexity Value $C_P = 3 + 8 = 11$

Complexity value for problem-2:

Create Table: $CT_v = 3$

Column value: $C_v = 3 \times \log_2(1+5) \approx 8$

Constraint value: $K_{wt} = 3 + 2 + 2 = 7$

Complexity Value $C_P = 3 + 8 + 7 = 18$

Complexity value for problem-3:

Create Table: $CT_v = 3$

Column value: $C_v = 3 \times \log_2(1+5) \approx 8$

Constraint value: $K_{wt} = 3 + 2 + 2 + 3 + 3 = 13$

Complexity Value $C_P = 3 + 8 + 13 = 24$

Complexity value for problem-4:

Create Table: CT_v = 3

Column value: $C_v = 3 \times \log_2(1+5) \approx 8$

Constraint value: K_{wt} = 3 + 2 + 2 + 3 + 5 + 2 = 17

Reference Table: t_w = 3

Complexity Value C_P = 3 + 8 + 17 + 3 = 31

Complexity value for problem-5:

Create Table: CT_v = 3

Column value: $C_v = 3 \times \log_2(1+5) \approx 8$

Constraint value: K_{wt} = 3 + 2 + 2 + 3 + 5 + 2 +3 = 20

Reference Table: t_w = 3

Complexity Value C_P = 3 + 8 + 20 + 3 = 34

Complexity value for problem-6:

Create Table: CT_v = 3

Column value: $C_v = 3 \times \log_2(1+5) \approx 8$

Constraint value: K_{wt} = 3 + 2 + 2 + 3 + 5 + 2 +3 + 4 = 24

Reference Table: t_w = 3

Complexity Value C_P = 3 + 8 + 24 + 3 = 38

Table 4.23: Complexity Value of SQL Create Statements

Problem No	Number of Column	Number of Constraint	Number of Reference Table	Complexity Value
1	5	0	0	11
2	5	3	0	18
4	5	5	0	24
4	5	8	1	31
5	5	9	1	34
6	5	11	1	38

4.6 Complexity Value of SQL Insert Statement

The complexity value of SQL Insert statement depends on the number of used column name, column value and sub-query. We have analyzed all clause using different parameters. Insert statement uses Select statement as sub-query. To calculate sub-query complexity, we have used the same formula as Select statement.

Table 4.24: SQL Insert Statement with Different Parameters

No.	SQL Create Statement
1	**`INSERT INTO contact (id, name, phoneNo) values (1, 'James', '017117')`**
2	`INSERT INTO contact (id, name, phoneNo) Values (1, 'James', '017117'), (1, 'James', '017117'), (1, 'James', '017117')`
3	`INSERT INTO contact (id, name, phoneNo) SELECT id, name, phoneNo from Employee where did = 5`

Complexity value for problem-1:

Complexity value of Insert: $I_v = 3$

Complexity value of Column Name: $C_w = 2 \times \log_2(1+3) = 4$

Complexity value of Column Name: $C_v = 2 \times \log_2(1+3) \approx 4$

Complexity Value $C_P = 3 + 4 + 4 = 11$

Complexity value for problem-2:

Complexity value of Insert: $I_v = 3$

Complexity value of Column Name: $C_w = 2 \times \log_2(1+3) = 4$

Complexity value of Column Name: $C_v = 2 \times \log_2(1+9) \approx 7$

Complexity value of Multirow: $C_v = 2 \times \log_2(1+2) \approx 3$

Complexity Value $C_P = 3 + 4 + 7 + 3 = 17$

Complexity value for problem-3:

Complexity value of Insert: $I_v = 3$

Complexity value of Column Name: $C_w = 2 \times \log_2(1+3) = 4$

Complexity value of sub-query: $SQV = 15$

Complexity Value $C_P = 3 + 4 + 15 = 22$

Table 4.25: Complexity Value of SQL Insert Statements

Problem No	Number of Mention Column Name	Number of used column value	Number of Multiple row	Complexity Value
1	3	3	0	11
2	3	3	0	17
3	3	3	2	22

4.7 Complexity Value of SQL Update Statement

The complexity value of SQL Update statement depends on the number of used column, condition and sub-query. We have analyzed all of those using different parameters. Update statement uses Select statement as sub-query. To calculate sub-query complexity, we have used the same formula as Select statement.

Table 4.26: SQL Update Statement with Different Parameters

No.	SQL Update Statement
1	**`UPDATE contact SET name = ‘Unknown’, phoneNo =‘017’`**
2	`UPDATE contact SET name = ‘Jack’, phoneNo =‘16319219742’ WHERE id = 3`
3	`UPDATE contact SET name = ‘Jack’, phoneNo =‘16319219742’ WHERE id in (Select empId from Employee where empID =121)`

Complexity value for problem-1:

Complexity value of Insert: U_v = 3

Complexity value of Column: $C_v = 2 \times \log_2(1+2) \approx 3$

Complexity value of Column: P_v = 2

Complexity Value C_P = 3 + 3 + 2 = 8

Complexity value for problem-2:

Complexity value of Insert: U_v = 3

Complexity value of Column: $C_v = 2 \times \log_2(1+2) \approx 3$

Complexity value of Where clause: w_v = 3

Complexity value of Column: P_v = 4

Complexity Value C_P = 3 + 3 + 3 + 4 = 12

Complexity value for problem-3:

Complexity value of Insert: U_v = 3

Complexity value of Column: $C_v = 2 \times \log_2(1+2) \approx 3$

Complexity value of Where clause: $w_v = 3$

Complexity value of Column: $P_v = 5$

Complexity value of sub-query: $SQV = 13$

Complexity Value $C_P = 3 + 3 + 3 + 5 + 13 = 26$

Table 4.27: Complexity Value of SQL Update Statements

Problem No	Number of Column	Number of Predicate	Use Sub-Query	Complexity Value
1	2	1	False	8
2	2	2	False	12
3	2	2	True	26

4.8 Complexity Value of SQL Delete Statement

The complexity value of SQL Delete statement depends on the optional where clause and sub-query. We have analyzed all of those using different parameters. Delete statement uses Select statement as sub-query. To calculate sub-query complexity, we have used the same formula as Select statement.

Table 4.28: SQL Delete Statement with Different Parameter

No.	SQL Delete Statement
1	`DELETE FROM contact`
2	`DELETE FROM contact WHERE id = 5`
3	`DELETE FROM contact WHERE id in (Select empId from Employee where name = ‘james’)`

Complexity value for problem-1:

Complexity value of Delete: $D_v = 2$

Complexity value of Table: $t_w = 3$

Complexity Value $C_P = 3 + 2 = 5$

Complexity value for problem-2:

Complexity value of Delete: $D_v = 2$

Complexity value of Table: $t_w = 3$

Complexity value of Where clause: $w_v = 3$

Complexity value of Column: $P_v = 2$

Complexity Value $C_P = 2 + 3 + 3 + 2 = 10$

Complexity value for problem-3:

Complexity value of Delete: $D_v = 2$

Complexity value of Table: $t_w = 3$

Complexity value of Where clause: $w_v = 3$

Complexity value of Column: $P_v = 3$

Complexity value of sub-query: $SQV = 13$

Complexity Value $C_P = 2 + 3 + 3 + 3 + 13 = 24$

Table 4.29: Complexity Value of SQL Delete Statements

Problem No	Number of Predicate	Use Sub-Query	Complexity Value
1	0	False	5
2	1	False	10
3	1	True	24

4.9 Comparing Complexity Value with Existing SQL-LES Systems

Table 4.30: Best Comparing Result with Existing SQL-LES Systems

Prob. No	Assignment No	Complexity Value	Complexity Level	Complexity Value from SQL-LES	Complexity Level from SQL-LES	Similarity
1	1	8.2	1	5	1	Ok
2	1	10.2	1	7	1	Ok
3	1	10.8439	1	9	1	Ok
4	1	14.6699	3	13	3	Ok
5	1	14.8	3	13	3	Ok
6	1	14.9699	3	14	3	Ok
7	1	14.9699	3	14	3	Ok
8	1	15.8	3	15	3	Ok
9	1	15.8	3	15	3	Ok

10	1	15.9699	3	16	3	Ok
11	1	16.9699	3	17	3	Ok
12	1	17.2699	3	17	3	Ok
13	1	17.5	4	17	3	Not OK
14	1	20.1399	4	19	4	Ok
15	1	19.4	4	18	4	OK
16	1	19.4	4	18	4	Ok
17	1	19.4	4	18	4	Ok
18	1	20.0908	5	19	5	Ok
19	1	20.3439	5	20	5	Ok
20	1	20.8699	5	19	5	Ok
21	2	30.8699	2	26	1	Not Ok
22	2	31.6304	2	30	2	Ok
23	2	31.8043	2	32	2	Ok
24	2	30.9742	2	32	2	Ok
25	2	30.9742	2	32	2	Ok
26	2	31.5138	2	31	2	Ok
27	2	30.7398	2	32	2	Ok
28	2	30.7398	2	31	2	Ok
29	2	33	2	34	2	Ok
30	2	33.1043	3	32	2	Not Ok
31	2	30.6893	2	32	2	Ok
32	2	34.8028	3	33	3	Ok
33	2	34.8028	3	33	3	Ok
34	2	33.5003	3	33	3	Ok
35	2	34.7605	3	33	2	Not Ok
36	2	32.3439	3	33	3	Ok
37	2	32.3439	3	34	3	Ok
38	2	32.3439	3	34	3	Ok
39	2	34.3304	3	29	2	Not Ok
40	2	34.3304	3	33	3	Ok
41	3	50.7439	1	48	1	Ok
42	3	55.0936	1	53	1	Ok

43	3	57.2888	2	55	2	Ok
44	3	54.8786	2	55	2	Ok
45	3	54.8786	2	55	2	Ok
46	3	53.0439	2	55	2	Ok
47	3	56.224	2	55	2	Ok
48	3	55.6893	2	56	2	Ok
49	3	55.6893	2	56	2	Ok
50	3	58.5264	2	58	2	Ok
51	3	58.5264	2	58	2	Ok
52	3	56.3439	2	57	2	Ok
53	3	56.3439	2	57	2	Ok
54	3	58.5801	2	58	2	Ok
55	3	60.6936	2	58	2	Ok
56	3	59.2098	2	58	2	Ok
57	3	60.9727	2	60	2	Ok
58	3	65.0145	2	65	2	Ok
59	3	63.8632	2	63	2	Ok
60	3	61.7047	2	61	2	Ok

4.10 Comparison Result with Existing SQL-LES Systems

We have compared our system with the existing SQL learning and evaluation system. We have tested our proposed model in two ways. First we have changed parameter value for all database items. Then we have changed our formula for sensitive parameter.

4.10.1 Comparison Result with Existing SQL-LES Systems by Changing Parameter Values using Table 4.1

To calculate complexity value, we have collected complexity value of individual database clause, function, predicate and others from three Database specialists. Table 4.1 has shown details about the all parameter value collected from different data specialists. Table 4.31 contrasts the performance comparison of new technique to that of previous methods. This model defined equivalence of problems maximum 83.3% of similarity compared to manually defined equivalence of problems.

Table 4.31: Comparing Complexity Value with Existing SQL-LES Systems by Changing Parameter Values

Test Case	Total Problem	No. of Similar Problem	No. of Dissimilar Problem	% of Similarity	% of Dissimilarity
Case-1	60	46	14	76.6%	23.3%
Case-2	60	46	14	76.6%	23.3%
Case-3	60	45	15	75%	25%
Case-4	60	50	10	83.3%	18.6%

We have changed the parameter value for test case one, two and three using collected parameter value from first, second and third database specialist.

Case-1: We have calculated complexity value using parameter value collected from first database expert one. In this case, complexity model define equivalence of problems 76.6% of similarity compared to manually defined equivalence of problems.

Case-2: This test has calculated complexity value using parameter value collected from second database expert. This case has also defined same result as test case-1.

Case-3: In this case, we have calculated complexity value using parameter value collected from third database expert. This case has defined equivalence of problems 75% of similarity compared to manually defined equivalence of problems.

Case-4: We have used the most approving value from database specialists to calculate complexity value. This case has defined equivalence of problems 83.3% of similarity compared to manually defined equivalence of problems.

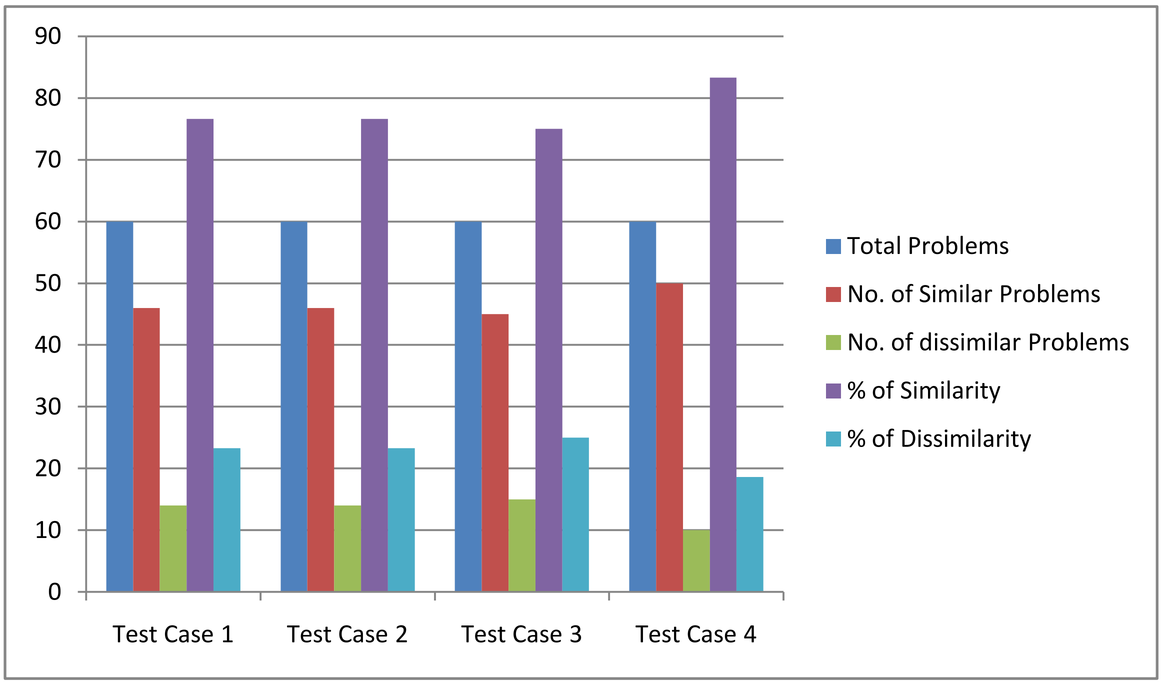

Fig. 4.15: Comparing Complexity Value with Existing SQL-LES Systems by Changing Parameter Values

4.10.2 Comparison Result with Existing SQL-LES Systems by Changing Formula and Parameter Values using Table 4.1

We have used different formula to calculate complexity value of a SQL problem. Predicate is the most sensitive parameter in SQL statements. We have changed formula for the most sensitive parameter using different parameter values from database specialists. Test case one, two, three and four has shown complexity value by changing formula for sensitive parameter. Test case four is used average parameter value from database specialists. Table 4.32 contrasts the performance comparison of new technique to that of previous methods. This model defined equivalence of problems maximum 91.6% of similarity compared to manually defined equivalence of problems.

Table 4.32: Comparing Complexity Value with Existing SQL-LES Systems by Changing Parameter Values and Formula

Test Case	Total Problem	No. of Similar Problem	No. of Dissimilar Problem	% of Similarity	% of Dissimilarity
Case-1	60	44	16	73.3%	26.6%
Case-2	60	46	14	76.6%	23.4%
Case-3	60	49	11	81.6%	18.3%
Case-4	60	55	5	91.6%	8.3%

We have changed formula for the most sensitive parameter using different parameter values collected from first, second and third database specialists. We have changed the formula to calculate the complexity value of predicate from –

$$P_v = \sum_{t=0}^{n} p_{wt}$$

to

$$P_v = \sum_{t=0}^{n} p_{wt} \times \log_2(1+n).$$

Case-1: We have calculated complexity value using parameter value collected from first database expert one. In this case, complexity model define equivalence of problems 73.3% of similarity compared to manually defined equivalence of problems.

Case-2: This test has calculated complexity value using parameter value collected from second database expert. This test has defined equivalence of problems 76.6% of similarity compared to manually defined equivalence of problems.

Case-3: In this case, we have calculated complexity value using parameter value collected from third database expert. In this case, complexity model has defined equivalence of problems 81.6% of similarity compared to manually defined equivalence of problems.

Case-4: We have used the most approving value from database specialists to calculate complexity value. This case has defined equivalence of problems 91.6% of similarity compared to manually defined equivalence of problems.

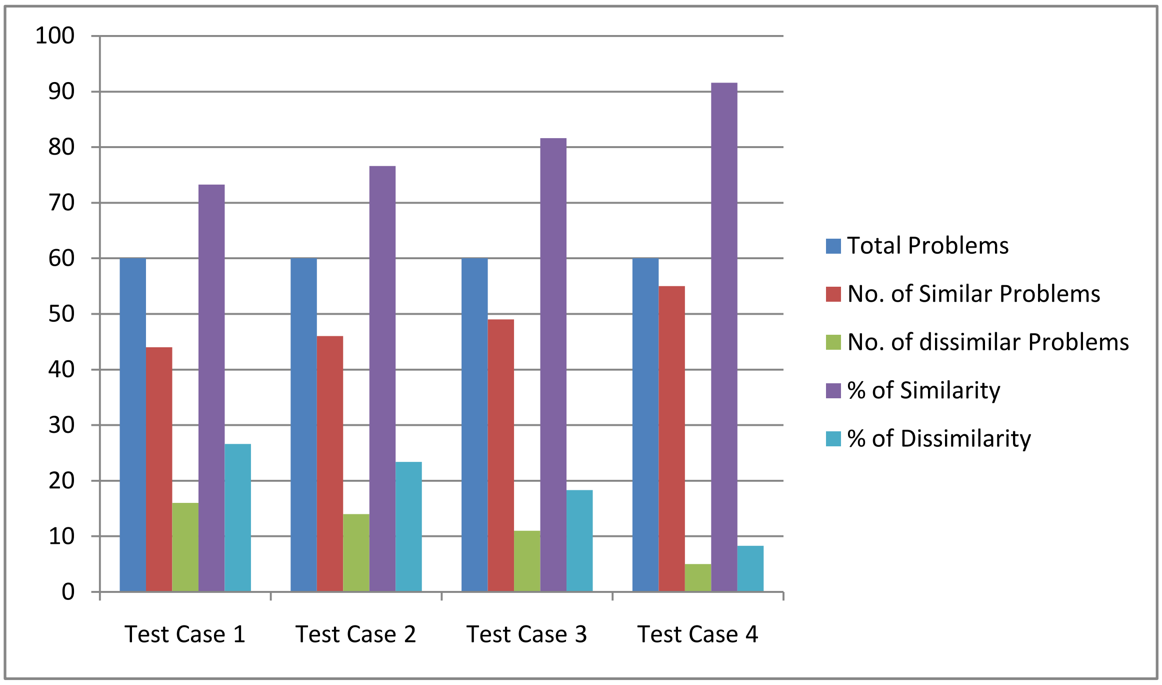

Fig. 4.16: Comparing Complexity Value with Existing SQL-LES Systems by Changing Parameter Values and Formula

4.10.3 Summary of the Comparison Results

The comparison result has shown that result of the proposed system is very close to manually define complexity of SQL problems. New system has defined problem complexity with 91.6% of similarity compared to manual systems using test case 4 of table 4.32. The minimum similarity is 73% using test case 1 of table 4.32. We have got the best result using average parameter value with changed formula. Our model behaves similar to the existing SQL-LES. We have included details about the test result in appendix section.

Chapter 5
Conclusion

The distribution of problems in problem-based education raises many issues for PBL implementation. Problems are the main element in problem-based learning. Student will learn through analyzing the assigned problem. They find out what need to learn, how they will get the resource, where they need to communicate to collect resource or information, how they will utilize their thinking power and others. By finishing those items, student will achieve their desired goal from learning. To evaluate student's performance in PBL session, it is important to distribute similar problems among the students. Complexity model will be very helpful to define equivalence of problems by analyzing complexity value.

In Problem-based Learning and Evaluation of SQL, students are assigned multiple assignments with a varying complexity. Existing SQL Learning and Evaluation systems assign the complexity values of SQL problems manually based on domain knowledge of the instructors. If the class size is large multiple instructors produce multiple assignments then it is difficult to have an equivalence of assignments. Students' performance sometime varies because of the dissimilarities of the assignments given by different instructors. At the same time, if the SQL question bank contains hundreds of questions, it is extremely difficult to obtain a global complexity value of each SQL problem to reuse the problems.

5.1 Contributions

Our contributions in this thesis can be described as follows:

- We have developed Complexity Model to find out the equivalence of problems using the complexity value of SQL problems. The equivalence of problems are same when whose problems fall into a given boundary of complexity value. To calculate the complexity value of a SQL problem, we have analyzed the problem in top-down fashion to find out the complexity of usages domain knowledge. To calculate complexity value, we have collect complexity value of individual database clause from three Database specialists.
- We have applied our proposed Complexity Model on existing problem based SQL learning and evaluation system question bank and found comparable result. This model defined equivalence of problems maximum 91% of similarity compared to

manually defined equivalence of problems. The minimum similarity is 73%. Our model behaves similar to the existing SQL-LES.

- Present system assigns the complexity value of SQL problems manually. Different instructors can assign different complexity values of the same problem. This will affect the student performance. The use of the complexity model will result uniform complexity values for all students.
- Manual assignment of complexity values increases the teacher workload. The application of our model will reduce the teacher workload in problem setting.

5.2 Future Research Direction

In this thesis we have found out the equivalence of SQL problems using the complexity value of different problems. To find out the complexity value of a SQL problem, we have parsed SQL query in sub-query and details. We can use the same concept for partial evaluation in problem based SQL learning and evaluation systems. In this work, we have find out the equivalence of SQL problems of Database course only. A generic Complexity Model can be developed to find out the equivalence of problems in problem based learning of other courses of engineering education by analyzing the problem in depth to find out the required domain and general knowledge.

References

[01]	Cindy E. Hmelo-Silver, "Problem-Based Learning: What and How Do Students Learn?," *Educational Psychology,* 2004, vol. 16, No. 3, pp. 235 – 266.
[02]	M. Qiu and L. Chen, "A problem-based learning approach to teaching an advanced software engineering course," in *Proceedings of 2010 Second International Workshop on Education Technology and Computer Science (ETCS),* 2010, pp. 252 – 255.
[03]	I. Richardson and Y. Delaney, "Problem based learning in the software engineering classroom," in *Proceedings of 22nd Conference on Software Engineering Education and Training, CSEET,* 2009, pp. 174 – 181.
[04]	R. Lacuesta, G. Palacios, and L. Fernandez, "Active learning through problem based learning methodology in engineering education," in *Proceedings of 2009 Frontiers in Education Conference,* 2009, pp. 1-6.
[05]	X. Qian, "A framework for designing problem-based learning environments," in *Proceedings of 2009 first International Workshop on Education Technology and Computer Science (ETCS),* 2009, vol. 2, pp. 16 – 20.
[06]	W. Hung, "The 3c3r model: a conceptual framework for designing in PBL," *The Interdisciplinary Journal of Problem-based Learning,* vol. 1, no. 1, pp. 55 – 77, 2006.
[07]	H. P. Yueh and W. J. Lin, "Developing a web-based environment in supporting students team-working and learning in a problem-based learning approach," in *Proceedings of Third IEEE International Conference on Creating, Connecting and Collaborating through Computing,* 2005, pp. 145 – 149.
[08]	L. Qiu and C.K Riesbeck, "Designing web-based interactive learning environments for problem-based learning," in Proceedings of Fifth IEEE International Conference on Advanced Learning Technologies, ICALT 2005. pp. 333 – 337.
[09]	R. Garcia-Robles, F. Diaz-del-Rio, S. Vicente-Diaz, and A. Linares-Barranco, "An e-learning standard approach for supporting pbl in computer engineering," *IEEE Journal of Education,* vol. 52, Issues: 3, pp. 328 – 339, 2009.
[10]	L. He, C. Wu, J. Yue, Z. Cai, and J. Liu, "Research & development of e-learning system for problem-based education," in *Proceedings of Education Technology and Computer Science, ETCS,* 2009, vol. 1, pp. 517 – 520.
[11]	L. Hoque, M. Islam, I. Hossain, and F. Ahmed, "Problem-Based e-Learning and Evaluation System for Database Design and Programming in SQL," *Interbational Journal of e-Education, e-Management and e-Learning,* 2012, vol. 2, no.6, pp. 537 – 542.
[12]	K. Hiekata, H. Yamato, P. Rojanakamolsan, and W. Oishi, "A framework for design engineering education with workflow-based e-learning system," *Journal of Software,* vol. 2, no. 4, pp. 88 – 5, Oct 2009.

[13]	O. Fontes, F. Neto, and A. Pontes, “A Multiagent System to Support Problem-based Learning,” *Scientific Research, Creative Education* 2011. vol.2, no.5, pp. 452 – 457
[14]	M. Asanok, P. Kitrakan, and C. Brahmawong, “Building a critical components for successful multimedia-based collaborative e-learning design framework,” *International Journal of the Computer, the Internet and Management*, vol. 16, no. SP3, pp. 37.1-37.10, 2008.
[15]	N. Vivekananthamoorthy, S. Sankar, R. Siva, and S. Sharmila, “An effective e-learning framework model - a case study,” in *Proceedings of 2009 7th International Conference on ICT and Knowledge Engineering*, 2009, pp. 8-14
[16]	L. Jantschi, S.D. Bolboaca, M.M. Marta, and A. Laszlo, “E-Learning and E-Evaluation: A Case Study,” in *Proceedings of 2008 Conference on Human System Interactions*, 2008, pp. 840 – 845.
[17]	M.A. Jabr and H.K. Al-Omari, “Design and implementation of e-learning management system using service oriented architecture,” in *Proceedings of World Academy of Science, Engineering and Technology*, vol. 64, pp. 59 – 64, 2010.
[18]	D. Tavangarian, M. Leypold, K. Nölting, and M. Röser, “Is e-learning the solution for individual learning?,” *Journal of e-learning*, vol. 2, Issue: 2, pp.273 – 280, 2004.
[19]	M. Wiggberg, “A method for analyzing learning outcomes in project courses,” in *Proceedings of 2010 IEEE Frontiers in Education Conference (FIE)*, 2010, pp. T4H-1 - T4H-2
[20]	R. Laughlin, C. Hatch, S. Silver, and B. Lee, “Groups Perform Better Than the Best Individuals on Letters-to-Numbers Problems: Effects of Group Size,” *Journal of Personality and Social Psychology*, 2006, vol. 90 ,no. 4, pp. 644 – 651
[21]	P. Lai and C. Tang , “Constraints Affecting the Implementation of problem-based learning (PBL) Strategy in University Courses,” in *Proceedings of the First Asia Pacific Conference on Problem Based Learning*, 1999, (pp. 49 – 54).
[22]	M. Archana, D. Sunil, G. Chitkara, and Madhu, “Designing problems for problem-based learning courses in analogue electronics: Cognitive and pedagogical issues,” *Australasian Journal of Engineering Education*, 2008, vol. 14 ,no. 2, pp. 33 – 41
[23]	E. Kaldoudi1, P. Bamidis, M. Papaioakeim, and V. Vargemezis, “Problem-Based Learning via Web 2.0 Technologies,” in *Proceedings of 21st IEEE International Symposium on Computer-Based Medical Systems*, 2008, pp. 391 – 396.
[24]	WIKIPEDEA, The Free Encyclopedia. *Problem-based learning* [Online]. Available: http://en.wikipedia.org/wiki/Problem-based_learning

Appendix

SQL Statements

No	SQL Statements
1.	`select * from lib_book`
2.	`Select sname,address from suppliers`
3.	`Select sid, sname,address from suppliers`
4.	`Select firstName,lastName,dateofBirth,district from sc_student where dateOfBirth<'01-jan-1989'`
5.	`select DId from Lib_Department where DName ='Mechanical Engineering'`
6.	`Select bookId from Lib1_book order by volume desc`
7.	`Select BookCopyId from Lib1_bookcopy order by PriceTaka desc`
8.	`Select AccessionNumber, AccessionDate from Lib1_bookcopy order by Binding desc`
9.	`Select bookid, title from Lib1_book order by YearOfPublication desc`
10.	`select DId, location from Lib_Department where DName ='Mechanical Engineering'`
11.	`Select max(pricetaka) from Lib1_bookcopy where YearOfprint = 2000`
12.	`Select count(*) from Lib1_bookcopy where YearOfprint = 2000`
13.	`Select courseNo, courseName from sc_course where courseNo like 'EEE%'`
14.	`Select avg(CGPA) from sc_student where slevel = 1 and term =2`
15.	`Select volume, count(*) from Lib1_book group by volume`
16.	`Select Binding, count(*) from Lib1_bookcopy group by Binding`
17.	`Select PlaceOfPublication, count(*) from Lib1_book group by PlaceOfPublication`
18.	`Select count(*) from Lib1_Publisher where PCountry = 'USA'`
19.	`Select studentId,firstName,lastName from sc_student where district in ('Chittangong', 'Rangpur', 'Dhaka')`
20.	`Select count(*) from Lib1_Book where BookGroup = 'Programming'`
21.	`select title,pname from Lib1_bookcopy,Lib1_book,Lib1_publisher where Lib1_bookcopy.bookid=Lib1_book.bookid and Lib1_book.pid=Lib1_publisher.pid and accessiondate>'01-JAN-2008' and accessiondate<'31-DEC-2008'`
22.	`select Title from Lib1_book,Lib1_bookdepartment,Lib1_department where Lib1_book.bookid=Lib1_bookdepartment.bookid and Lib1_bookdepartment.did=Lib1_department.did and dcodename='CSE'`
23.	`select title,afirstname,alastname from Lib1_book b,Lib1_bookauthor ba,Lib1_author a where b.bookid=ba.bookid and a.aid=ba.aid and b.yearofpublication=2006`
24.	`select title,afirstname,alastname from Lib1_book b,Lib1_bookauthor ba,Lib1_author a where b.bookid=ba.bookid and a.aid=ba.aid and b.yearofpublication>2006`
25.	`select title,afirstname,alastname from Lib1_book b,Lib1_bookauthor ba,Lib1_author a where b.bookid=ba.bookid and a.aid=ba.aid and b.purchaseDate>'01-JAN-2008'`
26.	`select title,afirstname,alastname from Lib1_book b,Lib1_bookauthor ba,Lib1_author a where b.bookid=ba.bookid and a.aid=ba.aid and b.pricebase between 200 and 500`
27.	`select ISBN,Title,BookGroup,DcodeName from Lib1_book,Lib1_bookdepartment,Lib1_department where Lib1_book.bookid=Lib1_bookdepartment.bookid and`

	Lib1_bookdepartment.did=Lib1_department.did
28.	select firstName, lastName, floor (months_between (sysdate,dateOfBirth)/12) from sc_student where slevel=3 and term=2 order by studentId desc
29.	select Title,Dcodename from Lib1_book,Lib1_bookdepartment,Lib1_department where Lib1_book.bookid=Lib1_bookdepartment.bookid and Lib1_bookdepartment.did=Lib1_department.did and dcodename = 'EEE' and purchaseDate > to_date('31-12-2007', 'dd-mm-yyyy')
30.	select title,afirstname,alastname from Lib1_book b,Lib1_bookauthor ba,Lib1_author a where b.bookid=ba.bookid and a.aid=ba.aid and b.PlaceOfPublication='USA'
31.	select slevel,term,avg(CGPA) from sc_student group by slevel,term order by slevel,term
32.	select afirstname,alastname from Lib1_book a,Lib1_book b,Lib1_author au,Lib1_bookauthor ba where a.bookid=ba.bookid and au.aid=ba.aid and a.yearofpublication=b.yearofpublication and b.title='Database'
33.	select afirstname,alastname from Lib1_book a,Lib1_book b,Lib1_author au,Lib1_bookauthor ba where a.bookid=ba.bookid and au.aid=ba.aid and a.placeofpublication=b.placeofpublication and b.title='Artificial Intelligence'
34.	select title from Lib1_book b,Lib1_bookauthor ba,Lib1_author a where b.bookid=ba.bookid and a.aid=ba.aid and a.alastname like 'K%'
35.	select afirstname,alastname from Lib1_book b,Lib1_bookauthor ba,Lib1_author a where b.bookid=ba.bookid and a.aid=ba.aid and b.BookKeywords='Structured Programming'
36.	select count(eid), avg(months_between(sysdate, birthdate)/12) from e1_employee where district like 'R%' and gender = 'M'
37.	select avg(months_between(sysdate, birthdate)/12), count(eid) from e1_employee where district like 'C%' and gender = 'F'
38.	select avg(months_between(sysdate, dateofbirth)/12), count(studentid) from sc_student where telephone like '011%' and sex = 'M'
39.	select DcodeName,DCodeNumber from Lib1_book,Lib1_bookdepartment,Lib1_department where Lib1_book.bookid=Lib1_bookdepartment.bookid and Lib1_bookdepartment.did=Lib1_department.did and placeofpublication like '%US%'
40.	select placeofpublication,yearofprint from Lib1_bookcopy,Lib1_book,Lib1_publisher where Lib1_bookcopy.bookid=Lib1_book.bookid and Lib1_book.pid=Lib1_publisher.pid and pemail like'%yahoo%'
41.	select title from Lib1_book where pricebase >(select min(avg(pricebase)) from Lib1_book group by yearofpublication) and yearofpublication > (select yearofpublication from Lib1_book where title='Database')
42.	select did, count(*) from Lib1_booking,Lib1_borrower where Lib1_booking.bid = Lib1_borrower.bid and did in (select did from Lib1_borrower, Lib1_booking, Lib1_book where Lib1_book.bookid = Lib1_booking.bookid and Lib1_booking.bid = Lib1_borrower.bid and title = 'Combinatorial Optimization') group by did
43.	select title,afirstname,alastname from lib_book b,lib_author a,lib_bookauthor ba where a.aid=ba.aid and b.bookid=ba.bookid and b.pricebase>400 and b.yearofpublication between (select yearofpublication from lib_book where title ='Database') and (select yearofpublication from lib_book where title ='Artificial Intelligence') order by title desc, afirstname desc
44.	select distinct pName, title from lib_bookcopy,lib_book,lib_publisher where lib_bookcopy.bookid=lib_book.bookid and

	lib_book.pid=lib_publisher.pid and yearofprint < some (select yearofpublication from lib_book where bookkeywords like '%Programming%') order by pname desc,title desc
45.	select distinct pName, title from lib_bookcopy,lib_book,lib_publisher where lib_bookcopy.bookid=lib_book.bookid and lib_book.pid=lib_publisher.pid and yearofpublication < (select min(yearofprint) from lib_book where bookkeywords like '%AI%') order by pname desc,title desc
46.	select title from lib_book where pricebase <(select max(avg(pricebase)) from lib_book group by yearofpublication) and purchasedate > (select purchasedate from lib_book where title='Programming with C')
47.	select distinct pName, title from Lib1_bookcopy,Lib1_book,Lib1_publisher where Lib1_bookcopy.bookid=Lib1_book.bookid and Lib1_book.pid=Lib1_publisher.pid and yearofprint < (select min(yearofpublication) from Lib1_book where bookkeywords like '%Programming%') order by pname desc,title desc
48.	select title from Lib1_book where pricebase >(select min(avg(pricebase)) from Lib1_book group by yearofpublication) and yearofpublication > (select yearofpublication from Lib1_book where title='Database')
49.	select did, count(*) from Lib1_booking,Lib1_borrower where Lib1_booking.bid = Lib1_borrower.bid and did in (select did from Lib1_borrower, Lib1_booking, Lib1_book where Lib1_book.bookid = Lib1_booking.bookid and Lib1_booking.bid = Lib1_borrower.bid and title = 'Combinatorial Optimization') group by did
50.	select title,afirstname,alastname from lib_book b,lib_author a,lib_bookauthor ba where a.aid=ba.aid and b.bookid=ba.bookid and b.pricebase>400 and b.yearofpublication between (select yearofpublication from lib_book where title ='Database') and (select yearofpublication from lib_book where title ='Artificial Intelligence') order by title desc, afirstname desc
51.	select distinct pName, title from lib_bookcopy,lib_book,lib_publisher where lib_bookcopy.bookid=lib_book.bookid and lib_book.pid=lib_publisher.pid and yearofprint < some (select yearofpublication from lib_book where bookkeywords like '%Programming%') order by pname desc,title desc
52.	select distinct pName, title from lib_bookcopy,lib_book,lib_publisher where lib_bookcopy.bookid=lib_book.bookid and lib_book.pid=lib_publisher.pid and yearofpublication < (select min(yearofprint) from lib_book where bookkeywords like '%AI%') order by pname desc,title desc
53.	select title from lib_book where pricebase <(select max(avg(pricebase)) from lib_book group by yearofpublication) and purchasedate > (select purchasedate from lib_book where title='Programming with C')
54.	select distinct pName, title from Lib1_bookcopy,Lib1_book,Lib1_publisher where Lib1_bookcopy.bookid=Lib1_book.bookid and Lib1_book.pid=Lib1_publisher.pid and yearofprint < (select min(yearofpublication) from Lib1_book where bookkeywords like '%Programming%') order by pname desc,title desc
55.	select title from lib_book where pricebase >(select min(avg(pricebase)) from lib_book group by yearofpublication) and placeofpublication in (select placeofpublication from lib_book where title='Programming with C')
56.	select title from Lib1_book where pricebase >(select

	`min(avg(pricebase)) from Lib1_book group by yearofpublication) and placeofpublication in (select placeofpublication from Lib1_book where title='Programming with C')`
57.	`select distinct title,pname,yearofprint from lib_bookcopy,lib_book,lib_publisher where lib_bookcopy.bookid=lib_book.bookid and lib_book.pid=lib_publisher.pid and pricetaka = (select max(pricetaka) from lib_bookcopy where pricetaka < (select max(pricetaka) from lib_bookcopy)) order by yearofprint,title,pname`
58.	`select distinct title,pname,yearofprint from Lib1_bookcopy,Lib1_book,Lib1_publisher where Lib1_bookcopy.bookid=Lib1_book.bookid and Lib1_book.pid=Lib1_publisher.pid and pricetaka = (select max(pricetaka) from Lib1_bookcopy where pricetaka < (select max(pricetaka) from Lib1_bookcopy)) order by yearofprint,title,pname`
59.	`select title from lib_book where pricebase >(select min(avg(pricebase)) from lib_book group by yearofpublication) and yearofpublication > (select yearofpublication from lib_book where title='Database') order by title desc`
60.	`select title from lib_book where pricebase >(select min(avg(pricebase)) from lib_book group by yearofpublication) and yearofpublication > (select yearofpublication from lib_book where title='Machine Learning') order by title desc`

Test Result

P. No	**C_p from SQL-LES**	**Test1**	**Test2**	**Test3**	**Test4**	**Test5**	**Test6**	**Test7**	**Test8**
1	5	8.6	8.6	7.6	8.2	8.6	8.6	7.6	8.2
2	7	10.6	10.6	9.6	10.2	10.6	10.6	9.6	10.2
3	9	11.2439	11.2439	10.2439	10.8439	11.2439	11.2439	10.2439	10.8439
4	13	14.7699	14.7699	14.7699	14.6699	14.7699	14.7699	14.7699	14.6699
5	13	14.9	14.9	14.9	14.8	14.9	14.9	14.9	14.8
6	14	15.0699	14.0699	16.0699	14.9699	15.0699	14.0699	16.0699	14.9699
7	14	15.0699	14.0699	16.0699	14.9699	15.0699	14.0699	16.0699	14.9699
8	15	15.9	14.9	16.9	15.8	15.9	14.9	16.9	15.8
9	15	15.9	14.9	16.9	15.8	15.9	14.9	16.9	15.8
10	16	16.0699	16.0699	16.0699	15.9699	16.0699	16.0699	16.0699	15.9699
11	17	16.7699	16.7699	16.7699	16.9699	16.7699	16.7699	16.7699	16.9699
12	17	17.7699	17.7699	16.7699	17.2699	17.7699	17.7699	16.7699	17.2699
13	17	18.6	17.6	**16.6**	17.5	18.6	17.6	16.6	17.5
14	19	18.7699	18.7699	18.7699	18.9699	19.9398	19.9398	19.9398	20.1399
15	18	19.9	20.9	17.9	19.4	19.9	20.9	17.9	19.4
16	18	19.9	20.9	17.9	19.4	19.9	20.9	17.9	19.4
17	18	19.9	20.9	17.9	19.4	19.9	20.9	17.9	19.4
18	19	20.8248	20.8248	19.2398	20.0908	20.8248	20.8248	19.2398	20.0908
19	20	19.8439	19.8439	18.8439	20.3439	19.8439	19.8439	18.8439	20.3439

20	19	21.0699	21.0699	20.0699	20.8699	21.0699	21.0699	20.0699	20.8699
21	26	28.2	28.2	29.2	28.4	30.3699	30.3699	32.3699	30.8699
22	30	28.6699	28.6699	29.6699	28.8699	31.2549	31.2549	32.8399	31.6304
23	32	28.8439	28.8439	29.8439	29.0439	31.4288	31.4288	33.0138	31.8043
24	32	28.8439	28.8439	29.8439	29.0439	30.5987	30.5987	32.1837	30.9742
25	32	28.8439	28.8439	29.8439	29.0439	30.5987	30.5987	32.1837	30.9742
26	31	28.8439	28.8439	29.8439	29.0439	31.0138	31.0138	33.0138	31.5138
27	32	29.3699	29.3699	30.3699	29.5699	30.5399	30.5399	31.5399	30.7398
28	31	28.0699	27.0699	31.0699	29.5699	29.2398	28.2398	32.2398	30.7398
29	34	29.5	29.5	30.5	29.7	32.5	32.5	34.5	33
30	32	30.1439	30.1439	31.1439	30.3439	32.7288	32.7288	34.3138	33.1043
31	32	30.4893	30.4893	30.4893	30.6893	30.4893	30.4893	30.4893	30.6893
32	32	30.6589	30.6589	31.6589	30.8589	34.3028	34.3028	36.3028	34.8028
33	33	30.6589	30.6589	31.6589	30.8589	34.3028	34.3028	36.3028	34.8028
34	32	32.3699	31.3699	31.3699	31.5699	34.1248	33.1248	33.7098	33.5003
35	33	31.5	31.5	32.5	32	34.085	34.085	35.6699	34.7605
36	33	32.5439	31.5439	29.5439	32.3439	32.5439	31.5439	29.5439	32.3439
37	34	32.5439	31.5439	29.5439	32.3439	32.5439	31.5439	29.5439	32.3439
38	34	32.5439	31.5439	29.5439	32.3439	32.5439	31.5439	29.5439	32.3439
39	29	33.2	32.2	32.2	32.4	34.9549	33.9549	34.5398	34.3304
40	33	33.2	32.2	32.2	32.4	34.9549	33.9549	34.5398	34.3304
41	48	50.4439	51.4439	49.4439	50.7439	50.4439	51.4439	49.4439	50.7439
42	53	50.5497	51.5497	49.5497	51.1497	54.1936	55.1936	54.1936	55.0936
43	55	52.6264	51.6264	55.6264	52.9264	56.5922	55.5922	60.9142	57.2888
44	55	52.6482	50.6482	52.6482	52.9482	54.4031	52.4031	54.9881	54.8786
45	55	52.6482	50.6482	52.6482	52.9482	54.4031	52.4031	54.9881	54.8786
46	55	52.4439	53.4439	51.4439	53.0439	52.4439	53.4439	51.4439	53.0439
47	55	53.8181	51.8181	53.8181	54.2936	55.573	53.573	56.158	56.224
48	56	54.6138	55.6138	52.6138	55.6893	54.6138	55.6138	52.6138	55.6893
49	56	54.6138	55.6138	52.6138	55.6893	54.6138	55.6138	52.6138	55.6893
50	58	54.6905	53.6905	55.6905	55.766	57.2755	56.2755	58.8604	58.5264
51	58	54.6905	53.6905	55.6905	55.766	57.2755	56.2755	58.8604	58.5264
52	57	55.7439	55.7439	56.7439	56.3439	55.7439	55.7439	56.7439	56.3439
53	57	55.7439	55.7439	56.7439	56.3439	55.7439	55.7439	56.7439	56.3439
54	58	57.5838	59.5838	53.9988	56.6498	59.3386	61.3386	56.3386	58.5801
55	58	55.8497	55.8497	56.8497	56.7497	59.4936	59.4936	61.4936	60.6936
56	58	56.9699	56.9699	52.9699	56.8699	59.8947	59.3098	54.7248	59.2098
57	60	58.6028	59.6028	59.6028	59.8028	59.7727	60.7727	60.7727	60.9727
58	65	60.7181	61.7181	62.1332	61.0842	64.473	65.473	66.473	65.0145

59	63	59.9028	60.9028	60.9028	61.1028	62.4878	63.4878	64.0727	63.8632
60	61	60.2292	62.2292	59.2292	61.7047	60.2292	62.2292	59.2292	61.7047