Graphical Expression of SQL Statements using Clamshell Diagram

Takehiko MURAKAWA*1, Masaru NAKAGAWA*1

Faculty of Systems Engineering, Wakayama University

Thinking process development diagram is a graphical expression from which readers can easily find not only the hierarchy of a given problem but the relationship between the problem and the solution. Although that has been developed as an idea creation support tool in the field of mechanical design, we referred to the restricted version as clamshell diagram to attempt to apply to other fields. In this paper we propose the framework for drawing the diagram of the SQL statement. The basic idea is to supply the hierarchical code fragments of a given SQL statement in the left side of the diagram and to put the meaning written in a natural language in the right. After describing the formulation, we show a few examples of SQL statements and the corresponding diagrams. Moreover we discuss a support system, namely a database of SQL statements, for understanding and writing down the statements.

キーワード: 情報の表現, 思考展開図, SQL, ソフトウェア管理

information representation, thinking process development diagram, SQL, software management

1 Introduction

In the Internet services such as electronic commerce and ticket reservation, it is typical that one or more servers employ some sort of database management system (DBMS) to hold the key data, while Web servers act as the wicket for responding users’ requests. Since the communication between Web servers and the DBMS using SQL is done within the server side, the users enjoy the service without regard to the database.
When constructing such a practical database system, the lightweight language like PHP (Hypertext Preprocessor) takes part of the logic and the Web servers’ behavior, but the SQL statement is often described as a string in the script file. That is why SQL is considered less serious from the viewpoint of source code maintenance.

Object/Relational (O/R) mapping is a mechanism to enable the developers to write the directions about record retrieval and manipulation. ActiveRecord used together with Ruby on Rails is the driving force. However we have to recognize that O/R mappers only supply the convenience of coding but the optimization of the SQL queries is untouched. For example, ActiveRecord leaves the method for executing any SQL statement directly, from which we can infer that it is a human work that makes the most efficient database access via SQL.

We investigated the application of thinking process development diagrams[1] (TPDDs) to lower process of software development. Although TPDDs have been developed for supporting mechanical design, the originators made the point that they are also useful in planning or organizing something in wider fields and in analyzing the phenomena or the existing products in detail. The authors agree to their assertion, and have been looking into the graphical expression for program understanding.

In this paper we propose the formalism for drawing the diagram of the SQL statement. After introducing the outline of symmetric clamshell diagram, we show the framework together with a few examples of SQL statements and the corresponding diagrams. Moreover we discuss a support system, namely a database of SQL statements, for understanding and writing down the statements in an appropriate way.

2 Clamshell Diagram

Thinking process development diagram is an idea generation support tool. In comparison with other tools such as KJ method, mind map, fishborn diagram (cause effect diagram) and TRIZ, TPDD is much more readable since you can find the whole-part relationships and the problem-solution relationship on a chart. So far as the authors know, idea generation supporting methods have been applied to the upper process of software design, development and improvement. Our approach is an attempt to assist the lower process.

There exist several variants of TPDD, but those diagrams were proposed mainly in the practical viewpoint whereas the structure was not investigated sufficiently. Therefore we focused on a simple version of TPDD and defined the data
structure to name it symmetric clamshell diagram (SCD) after its shape[2]. To sum up, an SCD consists of a pair of trees arranged axisymmetrically, where the roots of two trees are the both ends while each pair of corresponding leaves are connected in the middle. The left and the right tree means the problem and the solution respectively. Each node has a label containing some message, which is more general at the edges and more specific in the center. Although the number of level of a tree is unbounded, exactly 3 levels (6-layered diagram) seem to be convenient on screen.

We give a formal definition of the structure of SCD. An SCD is an undirected graph $G = (V, E)$ where $V$ and $E$ are the sets of vertices (nodes) and edges. The set $V$ is divided into two sets $V_l$ and $V_r$ while $E$ is split up into $E_l$, $E_r$, and $E_c$. Let $G_l = (V_l, E_l)$ and $G_r = (V_r, E_r)$, then they are isomorphic trees. Moreover, each leaf $v_l$ in $G_l$ must be connected to the corresponding leaf $f(v_l)$ in $G_r$ where $f$ is the isomorphic function of $V_l$ into $V_r$, that is $(v_l, f(v_l)) \in E_c$. Since the two subgraphs $G_l$ and $G_r$ are isomorphic, we can store in a computer the constitution together with the messages of all the nodes using a single tree by addressing every node appropriately. This implies that an SCD could be described by an XML document.

After investigating the structure of SCD, we drew the diagrams for a couple of pieces of open source software written in C including GNU Hello[2, 3]. The right half of the diagram shows a tree in which the labels are directory names, file names or code segments except comments. The left tree has the inverted connections so that the diagram may be line-symmetric, and the message on some nodes are the comments extracted from the files and the others are empty. The right-and-left allocation is due to a design concept of TPDD, or the belief that his or her thought should be expanded in terms of “what to do” and then converged by making clear “how to do” and integrating them. Using the drawn diagrams, we assessed the sufficiency of comments to make sure that the definition of an important function has more fulfilling comments.

### 3 Applying Clamshell Diagram to SQL Statement

Students of the authors’ department study SQL after mastering C and learning Java. Some of them have a resistance to this programming language since, say, the syntax is unprecedented for them. One of the goals in the database classes is that the students learn to write down the right queries by themselves.
Typically an SQL query consists of dozens of or a few hundred of bytes, much smaller than a source file written in popular procedural languages such as C. As far as the authors see, there is not a courteous comment attached to an SQL statement in practical system development, though those could be seen in answer books (for example, [4]) or for educational purpose. However it is not hard to partition a given SQL statement into the components (for example, FROM clause and WHERE clause in SELECT statement) and comment on the parts manually. Furthermore, it seems to be promising approach to automatically annotate such code segments using a number of pairs of the code segment and the comment stored in a database. That is how we are now considering the application of SQL statements to clamshell diagrams.

We will describe how to convert a given SQL statement into an SCD chart. In contrast to the way of drawing a conventional TPDD, our formalism arranges the code segments on the left half and the corresponding comments on the right. We adopt the reverse arrangement since the tree expression of an SQL statement on the left comes more naturally to SQL programmers who find the desired query by sending ones many times. Although out policy goes against the traditional way of TPDD, the author believe that this placement gains an advantage in a practical sense.

Imagine that you have a database of well-known jobs and their salaries, namely the table joblist which has the attributes title and salary. If you would like to know the jobs whose income is 50,000 (The currency unit is ignored.) or more, then execute “SELECT title FROM joblist WHERE salary >= 50000;”. We describe the diagram in Fig. 1. The provided statement is divided into the type of statement “SELECT”, the target attribute “title”, the FROM clause and the WHERE clause. The query is so simple that the labels are line-symmetric as well as the graph configuration. We consider that the values just after SELECT separated by commas should be expressed as the discrete nodes but complicated condition using AND or OR in the WHERE clause should be single since it is hard to analyze and segmentalize the expression.

Another example is a query including a subquery. To know the jobs which the worker gains the highest income in the list, “SELECT title FROM joblist WHERE salary = (SELECT MAX(salary) FROM joblist);” is the code. Figure 2 shows the SCD expression. Note that the subquery forms an SCD.

The variable $x$ in Fig. 2 is a parameter for grafting. When restoring the query and the description, we assign the one constituted by the subtree to $x$. The right
half of the diagram produces the direction “get the title from the joblist where the salary is equal to the maximum of the salary from the joblist”, where the label “get” on the root of the subquery is removed. What is substituted for each $x$ is unique on this diagram, but if there exists a message with more than one parameter, then the variable name should be specified along with the link.

Note that the diagram is about data expression or data structure, and we will have to choose more intuitive indication means.

4 Database of SQL Statements and SCD

To register and maintain a wide variety of SQL statements, the database is the cornerstone. In this section we are discussing the database for SQL statements. The issue includes (1) from what to collect and (2) how to store.

Collecting and selecting the actual SQL statements holds the key to the success of the constructing code repository. We are making a collection of them not only from books but from our development of database applications as well as the classes of database.

To draw an SCD from a given SQL statement, it would be impossible to find comment of the statement except for written in a book. We will annotate the code segments by hand initially, and subsequently implement the feature of the automatic comment. When each statement is moderately divided, a code segment is stored in the database together with the corresponding comment. In addition, the data for composing an SCD including the references to the code segments is preserved.

Many books for writing SQL statements have been published but the readers cannot invoke the code to verify it. We attempt to construct a database of verifiable SQL statements together with execution environments and data set to which the query is applied. Since the learners can see the result quickly, they will be able to know not only the syntax of SQL but good or bad SQL statements in a practical sense efficiently. It may be useful to those who brush up on SQL as well. Finally the database of SQL statements will be informative and helpful to the programmers of database manipulation from beginners to experts.

5 Conclusions

In this paper we have proposed the framework for drawing symmetric clamshell diagram provided an SQL statement. The configuration of the diagram with a pair of trees helps one to understand the whole and the parts of the query. In addition, the manipulation of the tree structure
such as substitution or grafting will produce another statement.

Future works include expressing SQL codes other than SELECT statements graphically and constructing a practical database of SQL statements together with their SCD configuration.

References


