SAVI: A new System for Advanced SQL Visualization

Maurizio Cembalo  
Dip. di Informatica  
“R.M. Capocelli”  
Università di Salerno  
Via Ponte don Melillo, I-84084  
Fisciano (SA), Italy  
maucem@dia.unisa.it

Alfredo De Santis  
Dip. di Informatica  
“R.M. Capocelli”  
Università di Salerno  
Via Ponte don Melillo, I-84084  
Fisciano (SA), Italy  
ads@dia.unisa.it

Ferraro Petrillo Umberto  
Dip. di Scienze Statistiche  
Università di Roma “Sapienza”  
P.le Aldo Moro 5, I-00185  
Rome, Italy  
umberto.ferraro@uniroma1.it

ABSTRACT
In this paper, we present SAVI, a new system for supporting the teaching and the understanding of the semantics of the SQL language. SAVI uses visualization to explain the way some of the SQL operators select and transform data from a target database. The contribution of our paper is two-fold. From a conceptual point of view, we improve the visualization approach provided by existing systems, in order to address the mental visualization problem faced by students when learning SQL. From a technological point of view, we leverage on emerging web technologies to develop a visualization infrastructure that can be seamlessly run on any standard HTML5-capable browser, without any need for an additional software or virtual machine installation.

Categories and Subject Descriptors
K.3.2 [Computer and Information Science Education]: Information systems education; H.2.8 [Database Management]: Database Applications; I.3.8 [Computer Graphics]: Applications

General Terms
Design, Experimentation

Keywords
SQL, Visualization, HTML5

1. INTRODUCTION
SQL (Structured Query Language) is a standard declarative language for defining, querying and manipulating relational databases (see [4]). The most common type of query performed in SQL is the SELECT query, which allows the extraction of data from an existing database.

Selection is typically done by “slicing” horizontally and/or vertically a target table, according to some user-provided conditions, so as to select only a subset of the information the table contains. However, SQL offers also many advanced functionalities, such as gathering data from multiple tables, deriving new information starting from the data available in existing tables, or running queries which recursively call other queries.

Despite its simple syntax, learning SQL and its underlying concepts may be difficult, especially when considering its more advanced functionalities. Several authors have tried to identify the reasons behind these difficulties. Sadiq et al. in [10] think that the main problem with learning SQL is its declarative nature. They observe that, when using procedural languages, complex problems can be approached by breaking them into simpler tasks to be solved in several steps. This approach cannot be followed with SQL, because in a complex query there are no intermediate steps to solve separately, but instead temporary sets of data which result from the execution of the different operators of the same query. So, as mentioned also by Russell et al. in [9], the resolution of a particular problem through SQL requires data-oriented problem-solving abilities. This analysis has also been done by Prior in [8]. He observes that the main problem faced by students when learning SQL concerns the mental visualization of the intermediary results coming from the execution of the operators of an SQL statement. Finally, Mitrovic in [6] remarks that it is hard to learn SQL by experimenting with an RDBMS because the possible error messages are limited only to syntax errors and do not provide any insight on the semantics of the operations being performed.

In this paper we present SAVI (i.e., SQL Advanced Visualization; it is also an Italian plural word denoting wise, sage), a new system that tries to improve on existing systems that address the mental visualization problem by using visualization. In our system, an input provided selection query is explained by representing the behavior of its operators in their natural evaluation order, when run against a target database. For each operator, the system visualizes the input data sets, the way the operator works, and the output data sets, with an emphasis on the way input data is transformed into output data. Moreover, the user has the possibility to interactively play back and forth several times one or more steps of the visualization, as desired. SAVI is primarily intended to be a support tool for the teacher, to integrate the traditional way of teaching SQL using the provided visual metaphors. Moreover, it can also be used by students to advance their understanding of SQL. The system has been implemented in Java, as a client-server distributed application, and leverages on emerging web technologies to...
run on any standard HTML5 browser without requiring any additional plug-in installation.

2. RELATED WORK

In the past years, several approaches have been proposed in order to simplify the teaching and the understanding of SQL with the help of visualization. These systems use visualization to describe and to explain the way an SQL query interacts with a database or with the physical or the logical components of the underlying database management system (DBMS). They are mainly intended to be used by a teacher, during a class, as a support tool, but they may be used as well by students to improve or experiment with their comprehension of SQL. The first visualization system we are aware of is eSQL, presented by Kearns [5] in 1997. This tool executes an input SQL SELECT statement and, apart from returning the result of the query, it visualizes, step-by-step, the way the query selects the output data. Here, the emphasis is put on visualizing the behavior of the operators of a query.

Allenstein et al. presented in [1] an alternative query simulation system that allows students to visualize the steps involved in processing SQL queries. Students can execute a set of predefined statements or connect to a live database and execute any desired command. This system differs from eSQL because the emphasis, here, is not put on explaining the way the commands of the SQL language can be used to extract information from a database. Instead, the focus is on visualizing which physical components of the DBMS are involved in the execution of a query, and their interactions. A similar approach has been followed by Murray et al. in [7]. Their system includes a set of software animations that support the teaching of database concepts such as database design, stored procedures and SQL queries. The students can use this system to learn basic and advanced SQL commands, by assembling statements and running them with the help of an interactive guide. Here, visualization is used to describe the input and the output of a SQL statement, without no emphasis on the way the statement selects data from the input database.

Visualization is also often used by SQL assessment systems. These typically assume that the user has already a basic knowledge of SQL and challenge him by posing a sequence of data selection problems that must be solved by formulating a proper query. Visualization, here, is used to display the content of the database the user will interact with, and to compare the output of these queries with the expected output. One of the first assessment systems to be presented to the scientific community has been SQL-Tutor, a system presented by Mitrovic in [6] and designed as a problem-solving environment, complementing traditional lessons, which attempts to provide intelligent feedback to the students who use it. The system records the state of learning of each student, and challenge them by asking them to design queries whose difficulty is proportional to their knowledge. When the student inputs a solution, SQL-Tutor analyzes it, searching for errors, notifies the student about the correctness of the provided solution and updates his state of learning consequently. AsseSQL [8] and SQLator [10] are also two interactive SQL assessment systems usable via a web browser. Finally, SQLify [3] is a tool that adds to the aforementioned systems the possibility, for a teacher, to perform a peer review of assignment work done by students.

It is interesting to observe that most of these systems are focused on presenting the details of the physical execution of a query and on the way the DBMS manages it. Instead, little attention is paid to the problem of explaining the semantics of SQL. The only exception seems to be eSQL, which, instead, pays more attention to the mental visualization problem, by explicitly representing the way SQL operators interact with input data during the execution of a query, even when very simple visualization metaphors are used. This system, which is no longer available, was coded as a single monolithic application, including a database management system.

3. OUR PROPOSAL

In this paper we present a new system\(^1\), falling in the category of the SQL visualization systems, designed for supporting the teaching of the semantics of SQL. Our system works by offering the user the possibility to connect to an external database and issue SQL data selection queries. If the provided query has no errors, it is visualized by progressively displaying a set of animations representing the way the different operators existing in the query interact with the input database.

Our contribution to the SQL visualization research field, with respect to existing systems, is two-fold. From a conceptual point of view, we improve and extend the visualization approach of eSQL, by introducing \textit{reversible animations} and putting more emphasis on the data-visualization aspects. From a technological point of view, we leverage on emerging web technologies to develop a visualization infrastructure that can be seamlessly run on any standard HTML5-capable browser, without any need for an additional software or virtual machine installation. These two contributions are discussed in detail in the rest of this section.

3.1 Reversible Animations

The only system existing so far for teaching the semantics of SQL, eSQL, describes a query by breaking it into several simple steps. Each step is visualized by displaying its input table and the resulting output table. Simple forms of animation, such as highlighting a row or a column, are used to emphasize the portion of data being processed by the query at each step. This basic visualization is integrated by a text field, displayed above each table and used to describe what is going on through standard explanatory messages.

The visualization approach followed by our system is, apparently\(^2\), similar to eSQL. The representation of a query is broken into several \textit{stages}, each corresponding to the resolution and to the execution of one of the elementary operations required by that query. Each stage presents to the user the set of tables involved in that operation, displays a graphical transition describing the behavior of the operation, and returns the resulting output table. All the rows, columns or cells being targeted by an operation are emphasized through the use of colors.

The main difference is in the way we represent the execution of each step. The visualizations generated by eSQL are essential and are more focused on displaying the sequence of intermediate tables deriving from the execution of a query.

\(^1\) A copy of the system is available upon request.

\(^2\) A direct experimental comparison between the two systems was not possible because eSQL was no longer available.
Instead, in our system, more emphasis is put on visualizing and explaining the way the SQL operator works and the way the information is transformed.

To this end, our system uses reversible animations (i.e., graphical transitions that can be played back and forth), which recalls the path-transition paradigm presented by Stasko in [11], to explicitly explain what an SQL operator does, whereas eSQL leaves this task to the intuition of the student. We believe that this approach is more effective, because it is able to better develop in the student the data-oriented problem-solving ability described by Russell et al. in [9].

For example, the JOIN operator, that is in charge of merging together the lines of two tables satisfying a user-provided condition, is represented in our system by displaying, at the same time, both the two input tables and the output table. The animation shows the rows of the two tables being currently checked. At each step, if the two rows satisfy the user’s condition, they are cloned and their copies are merged and moved in the output table. The same operation is visualized in eSQL by just displaying the two input tables and, then, the table resulting from the JOIN.

3.2 Web-based Visualizations

All the SQL teaching systems presented so far have been implemented as desktop applications (e.g., eSQL, written in C and Tcl/Tk and running on X-11 based systems) or as Java applets (e.g., [7, 2]). Indeed, these systems are mainly intended to be run by a teacher during the lesson. It is as well common for the teacher to use, to this end, a calculator where he or she often has limited administrative rights and/or has to deal with software-related issues.

We have overcome these problems by coding our system using only standard web technologies. This makes it possible to access our system using a standard web browser, without any further installation, deployment or administrative issues. However, an SQL visualization system has some strong requirements which are not easily satisfied in a web-browser only environment. First, there is a need to use an SQL parser and an SQL engine. These are required to analyze the structure of the queries provided by a user, in order to choose which visualizations to trigger, and to execute them against a target database, in order to determine which portion of data should be involved in the visualization. Second, there is a need to be able to draw arbitrary graphics and sophisticated animations on the browser screen, such as those required in our visualization.

Despite the complexity reached by current widespread web technologies, these two requirements still seem to be difficult to satisfy. As far as we know, for example, there is no off-the-shelf Javascript-based SQL parser available today. Similarly, the primitives available with the Javascript language would allow us to draw our visualizations, but at the expense of lot of effort. These problems have been solved by implementing our system as a client-server application. The server-side, implemented in Java, includes an SQL parser and an SQL engine, and carries out all the activities related to the parsing and the step-by-step execution of SQL queries. The client-side, implemented in Java and converted in Javascript, uses an animation library we coded that leverages on HTML5 primitives and allows the rendering of SQL visualizations.

Figure 1 illustrates the web interface of the system proposed. In the top of the interface we have the area where tables and animations are displayed. In the lower part, the user can enter the SQL query to be submitted to the system.

4. ARCHITECTURE

We have implemented our system as a client-server distributed application using the Google Web Toolkit framework. This framework allows to create rich internet applications by writing the client code in Java, thus gaining all the advantages of this language, while generating Javascript code that can be executed in any browser. The server part of our system runs in a standard Java environment and includes a relational database management system and an SQL parser. The client part of our system is in charge of rendering the user interface, generating the visualization and dealing with interactions coming from the user. The overall architecture is illustrated in Figure 2.

Figure 2: The architecture of our system

On the client side, after loading the application, the user inputs the address of the database to use. This input is sent to the server that, in turn, imports the database and analyzes its content. The list of available tables is returned to the client, together with the details about the structure of each table. Then, the user may interact with a command window to input the text of the query to be visualized.

The SQL queries provided by the user are sent to the server, where are analyzed by the Parser module. If any error is found, it is reported to the client application. Oth-
erwise, the analysis ends by establishing the set of steps required for the execution of that query. Each step gives rise to a different reversible animation, as described by the object Stage. Thus, the preliminary analysis is used to instantiate an array of Stage objects, and to determine which kind of animations they will trigger. After the analysis ends, the query is sent to the Director module, where it is run in a step-by-step way. The intermediary results of this execution are used by the Director module to fill the array of Stage objects with details about which cells, rows or columns of the target tables have to be involved in the animation and which way they will be transformed. At the end of this phase, the sequence of Stage objects that will begin the visualization will be returned to the client. The visualization is rendered using a sequence of tabbed panels, each containing a canvas displaying a different stage of the visualization. At the beginning of the visualization, the focus is put on the first stage. At any time, the user has the possibility to replay the current stage, skip to the next stage or return to a previous stage.

4.1 Graphical Animations with HTML5

One of the most interesting features of the HTML5 language is the Canvas element (see [12]). It allows the programmatic rendering of bidimensional shapes in an HTML5 page using the Javascript language. In its current formulation, this element is still relatively simple yet powerful. It exposes functions for drawing basic and complex geometric shapes and text, composing different shapes together through image processing operators, and reacting to user events.

Despite these functions, there were two important requirements for our system that were not met by the standard Canvas implementation. First, the available drawing primitives are too low-level. The rendering of a complex scene requires issuing the code required to draw each single shape in that scene. The drawing of a table, e.g., would require the developer to keep track of the graphical status of each single cell of the table, including its content. This problem becomes more serious if we consider that, in our case, the choreography of an animation is decided on the server, while its visualization occurs on the client. Therefore, by describing a complex animation using low level primitives, the amount of data to exchange between server and client may compromise the performance of the system. Second, there is no explicit support for animations.

We solved the first problem by developing a hierarchy of classes representing, at different levels of detail, the content of a table. The top-level class is Table; it is defined as a collection of Row objects and of Column objects. These two classes are, in turn, defined as containers of Cell objects. This last class is in charge of maintaining the state of a single cell of a table, including the data it contains, and of taking care of all that is needed to draw it on the screen. This organization resembles that of a scene graph, since all the transformations applied to an object are recursively propagated to all the objects it contains. Moreover, it allows the user to simply and seamlessly manage the visualization of a table at different levels of details, according to the operation being described.

Support for animations has been implemented by delegating to all graphical objects the responsibility to draw themselves on an output canvas. When a new graphical object, e.g., an instance of the Table class, is created, it is registered to the SQLCanvas. This class is in charge of setting up, maintaining and updating the animation of a graphical scene on a canvas. After being instantiated, it uses the standard setTimeout method to suspend, on a regular basis, the current execution of the client application, and to ask all the registered objects to update the status of their animation.

5. SQL VISUALIZATION

The SQL language may be used for several different purposes. It is commonly used to select, update, delete or insert data in an existing database. However, it is also used for manipulating databases, tables, etc., or for establishing or revoking permissions on the objects of a database.

In our system, we decided to restrict our attention to only selection queries (i.e., queries that extract data from an existing database), as implemented by the SELECT operator, as these are typically the most relevant when teaching SQL. In the following paragraphs, we describe briefly the operators supported by our system, and their visualization, listed according to the order used in their resolution and visualization. Each operator is visualized in a different stage, and the table resulting from the execution of one operator is used as input for the execution of the subsequent operator.

FROM . . . . JOIN.

The FROM operator determines the table that will be used to carry out the selection. Multiple tables may be combined through the use of the JOIN operator. This second operator works by merging the rows of two or more tables according to a user-provided join condition (usually, an equation involving some of the attributes of the included tables). If no condition is provided, we have a Cartesian product, i.e., each row of each table will be combined with all the rows of the other tables. The JOIN operator is described by visualizing the content of all the tables involved in the operation. The columns corresponding to the attributes involved in the JOIN condition are highlighted. The animation displays one arrow for each table, pointing at the rows currently considered. If the set of rows currently considered obeys the JOIN condition, the corresponding rows are cloned and their copies are merged together in the output table (see Figure 3). Then, the animation proceeds by selecting new rows.

WHERE.

This operator dictates the condition that has to be satisfied for a row to be included in the selection. At the beginning of the stage, the input table is drawn on the canvas. Then, an arrow begins to scan all the rows of the table. For each row, the columns corresponding to the attributes involved in the WHERE condition are highlighted. If the condition is satisfied, the corresponding row is cloned and put in the output table. The animation ends when all the rows have been considered.

GROUP BY.

This operator groups the rows of a table in one or more subsets of rows, where each subset contains all the rows of the original table that have identical values on one or more target attributes chosen at input. At the beginning of the stage, the input table is drawn on the canvas. The columns corresponding to the attributes used for the grouping are
Figure 3: Visualization of the JOIN operator. The rows of the tables Departments and Employees sharing the same value of the attribute Dept_ID are cloned, merged and moved to the output table.

highlighted. Then, an arrow begins to scan all the rows of the table. For each row, if the values it assumes on the grouping attributes have been read for the first time, then the row is cloned and moved onto a different part of the canvas, where it will form a new subset (see, e.g., Figure 4). Instead, if the values have already been read in a previous row, the cloned row is moved into the already existing subset.

HAVING.

This operator dictates the condition that has to be satisfied for a row of a table that has been subject to a GROUP BY operator to be included in the selection. The animation is identical to that of the WHERE operator.

Aggregate operators (e.g., SUM, AVG, MAX).

These operators perform a calculation on a set of values belonging to the same column and return a single value. They are often used in conjunction with the GROUP BY operator. In this case, the aggregate operators are evaluated on each single subset returned by the grouping operations. The visualization of these operators works by displaying, at first, the input table (or tables, if we have used the GROUP BY operator), with the column that will be used for evaluating the aggregate operator highlighted. Then, a new column is added to each table having, as header, the name of the operator and, as content, the result of the operator.

SELECT.

This operator chooses the columns to be included in the output of the selection. The list, usually denoted as “target list”, may include also columns processed through aggregate operators. The animation of this operator displays, initially, the input table. The columns included in the target list are highlighted and cloned. The resulting columns are moved to a new table, i.e., the overall output of the query.

6. FUTURE DIRECTIONS

In this paper we have presented SAVI, a new system that uses visualization for supporting the teaching and the understanding of the SQL language. SAVI features several differences with respect to existing systems. First, it uses reversible animations to explain in detail how the operators of a query select and/or transform data from an input database. This approach helps users to overcome the mental visualization problem, concerning the difficulty in understanding the way the different operators of an SQL complex query interact with a database up to the final result. Second, it is flexible. It gives the user the possibility to connect to an SQL-compliant database, run any query and interactively browse the resulting visualization. Third, it uses emerging standard web technologies to deliver a tool that can be run within any standard web browser, without the need for any additional software installation.

As a future direction, we are planning to perform an experiment to evaluate the effectiveness of our system when used, during a class, to support the explanation of some of the topics related to SQL queries. Another direction to investigate is the visualization of SQL statements running on large tables. In its current formulation, our system is able to visualize in an effective way only queries running on tables having a limited number of rows. However, the explanation of some of the most complex topics related to SQL queries may require the use of much larger tables.

There are several possible solutions to this problem. For example, it would be interesting to experiment with an alternative visualization for database tables, not based on the literal representation of the contents of the table, but relying on some sort of visual metaphor requiring less space to be drawn. Another possibility would be to use a zoomable user interface to represent tables in their entirety and, then, focus only on the portion of a table involved in the current operation. All these solutions would require, anyway, a further strong development of the HTML5-based visualization library we used in our system.
7. REFERENCES


