Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 10 (2012) 752 – 757

Procedia Computer Science

The 9th International Conference on Mobile Web Information Systems (MobiWIS)

Authoring relational queries on the mobile devices

Adele Hedrick[a], Ken Q. Pu[b]

Faculty of Science
University of Ontario Institute of Technology
2000 Simcoe Street North, Oshawa, Ontario, Canada

[a]*adele.borer@mycampus.uoit.ca*
[b]*ken.pu@uoit.ca*

Abstract

In this paper, we present the design and implementation of a graphical user interface to interact with a relational database management system on touch screen based mobile devices. Our system allows users to author arbitrary graphical relational queries on the touch screen using a collection of on-screen widgets and gestures. The user can interact, introspect and integrate pieces of sub-queries on the mobile device. The system dynamically performs type checking during the authoring, so the user receives *immediate* visual feedback in case of semantic error in the resulting query.

We have deployed our system on several Android devices ranging from the 4" smart phones to 7" and 10" tablets. Our usability study shows that our query interface provides a significant improvement in the user experience when querying a relational data model. The system allows fast authoring speed of complex queries, and user friendly experience.

Keywords: touch screen, mobile, user interface, relational database, relational query

1. Introduction and Motivation

In the first decade of the new millennium, we have witnessed a number of disruptive technological innovations and phenomena. Mobile computing has become the fastest growing field in industry and academia alike. The rapidly evolving ecosystem of touch screen based mobile devices (smart-phones and tablets) together with the advancements in cellular and wireless networking have truly made high performance computing pervasive. Aside from the compact and energy efficient processor design, mobile devices have also revolutionized the way users interact with computers. Touch screen is now the de-facto standard input device [1, 2, 3], and gestures (with multiple touch points) are becoming commonly used [4, 5, 6].

The past decade has also been marked by an explosion of personal information. From online sources, such as financial institutions, social networks, push-based Web content provider, and mobile messaging services, we are faced with an unprecedented volume of *personal information*. With the abundance of information, and the computing resources offered by mobile devices, there is a distinct opportunity of enabling end users to perform deep data analytics of their personal data on their own mobile devices on the go.

doi:10.1016/j.procs.2012.06.096

Traditional data analytics have been exclusively performed by databases. Most mobile operating systems are equipped with an embedded relational database management system (RDBMS) (e.g. SQLite for iOS and Android). The problem is that the end users are unable to harness the power of a query language (e.g. SQL) for RDBMS. Currently, many domain specific mobile applications have been developed to close the gap between the underlying RDBMS and the analytic needs of the end user.

2. Related Work

With the popularity of touch screen devices, gesture based UI design has received much attention [1, 2]. Ruiz el al [6] studied the best practices in mapping motion based gestures to user defined commands. Lu and Li [5] introduced *gesture avatar* for user defined gestures. Li [4] introduced *gesture search* to replace traditional keyboard based search. Widgor [3] showed on screen visual feedback, known as *Ripples*, enhances the user experience when interacting with a touch screen.

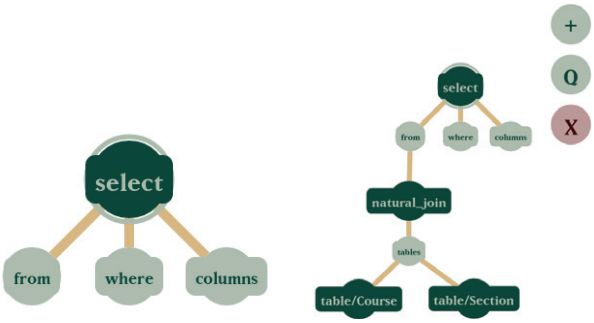
Unlike above gesture based methods, our problem of SQL query construction requires to produce artifacts that must satisfy strict syntactic grammar (defined by the SQL standard). In addition to flexible gesture and touch driven input, we also need to enforce structural correctness of the intermediate queries. Similar to the work in [3], we use visual feedback to provide suggestions and error reporting to the user.

Kleek et al [7] proposed a tool for personal information management , while Oleksik [8] investigated information management through tagging. Both can be thought of as specific cases of SQL query construction, but their scopes are much more limited than our goal of generic querying of relational databases.

In the database community, several graphical database query languages have been proposed [9, 10, 11]. Due to the evolution in display and touch screen, these graphical query languages cannot be implemented on small mobile displays with touch screen interfaces.

3. User Interface Design

3.1. Graphical Representation of SQL



(a) Emtpy Select Operator (b) A Complete Query

Fig. 1. Select Operator

To graphically represent SQL we decided to use a tree structure where there are two types of nodes; operators, and handles. The operators include; select, join, natural join, table, column, and, or, etc. The handles represent parameters for an operator, for example the select operator (Fig. 1a), would include the following handles; from, where, and columns. An example of a small complete query is shown in Fig. 1b, where two tables have been joined together using the natural join operator.

To simplify building a query, the dark nodes can only connect to light nodes, and the light nodes can only connect to dark nodes.

The node that starts the query is called the root node, and this is denoted by the node being circled by light green. The *select*node in both Figure 1a and Figure 1b are selected as the root. This functionality allows a user to have multiple queries on the canvas and can switch the query that will be executed.

3.2. User Interaction

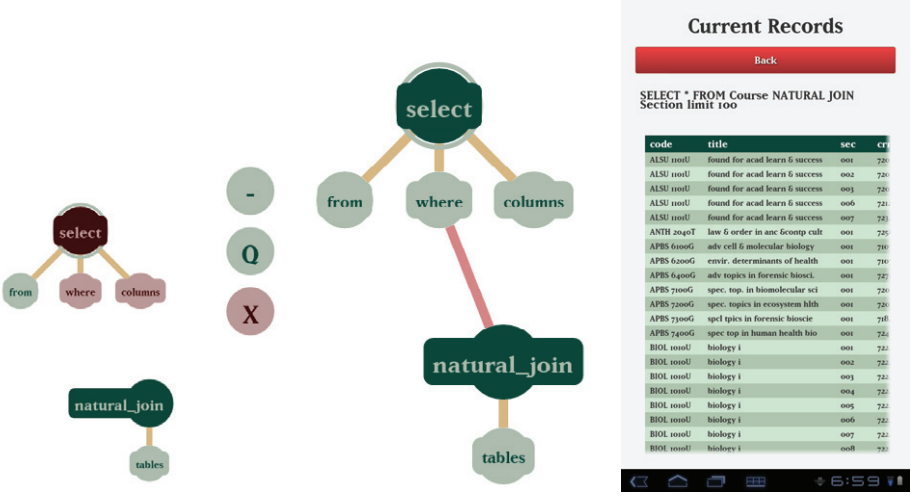
3.2.1. Adding Nodes

(a) Adding operator nodes (b) Adding tables and columns

Fig. 2. Menus

The user is able to add nodes to the canvas by tapping the plus button (shown in Figure 1b) which will bring up the main add node menu (shown in Figure 2a). The three buttons under value bring up an additional menu. If the user wishes to insert a table, they tap the table button on the main add node menu (Figure 2a), a new menu (Figure 2b) is then displayed with the table drop down being populated with the tables in the local database. The user then has to press insert to complete the selection, as with the selection of a column or constant. If the user was to press any of the other selections on the main add node screen (Figure 2b) they would automatically return to the canvas.

3.2.2. Removing Nodes



(a) Removing an operator (b) Connections that violate the type systems are highlighted. (c) Final query result

Fig. 3. Identifying Errors and Viewing Results

When a node is being dragged or held by the user, the plus button becomes a minus button, and if the user drags the held node over the minus button, the node is then removed from the canvas. In Figure 3a the *natural join* node is currently being held, and the plus button has been turned into a minus sign.

3.2.3. Making and Breaking Connections

The user is able to connect an operator to a handle by simply dragging the operator over the handle, which will then creates a connection made visible by the light brown lines. To break a connection, the user presses and holds on the operator they wish to disconnect from the handle.

3.2.4. Moving Nodes

The user is able to move the nodes by pressing on a node and dragging it to the location of their choice. Moving a node will also move all the children nodes in relation to it's parent. Connections are not broken when nodes are being moved, and are instead stretched with the nodes.

3.2.5. Canvas Operations

The canvas itself has two gestures that it recognizes; the pan gesture which allows the user to drag the canvas around to move all the nodes at once, and the fling gesture which will clear the canvas completely and then add a single empty select node as shown in Figure 1a.

3.2.6. Setting the Root

The root node (as mentioned in Graphical Representation of SQL) is selected by the user demonstrating a *long press* on a node with no parent.

3.2.7. Displaying Records

When the user wants to execute a query, they must have a root node selected and then tap the "Q" (query) button as shown in Figure 1b and Figure 3a. Once the query is submitted they will be brought to the *Current Records* screen (Figure 3c). This screen displays the generated SQL as well as the records in a table with a distinct header row, and alternating row colours for the content. Since the application can be used on many different screen sizes, the records table has the ability to scroll vertically and horizontally.

3.3. User Feedback

User feedback is a very important aspect of human-computer interaction, and we incorporated user feedback into the design of the GUI to assist the user in creating correct queries. In Figure 3a, the user is holding onto the *natural join* node, which not only moves the label further to the right to help the user remember what they are holding, but also identifies what nodes on the canvas the held node should not be attached to by colouring unacceptable nodes red.

The user still has the freedom to connect nodes together even if the multiplicity value is already satisfied or the type checking deems the connection to be an unacceptable one, but bad connections are denoted by a red line rather than a brown line as shown in Figure 3b.

4. Implementation

We have implemented the proposed SQL authoring application for the Android operating system. The Android architecture naturally decouples the application into a collection of activities as shown in Figure 4.

The Android 2D graphics API is used to implement the rendering and animation of the query authoring widgets such as operator nodes and operator connections. Our application support both accessing remote as well as on-device relational databases. Remote database access is implemented using a JSON based network protocol, while local database access utilizes the Android SQLite API.

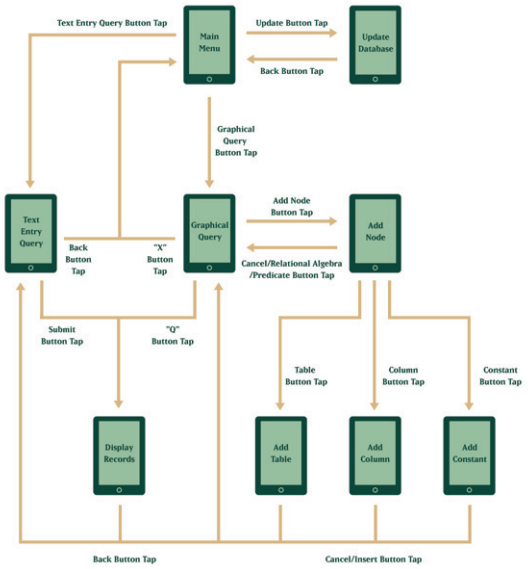
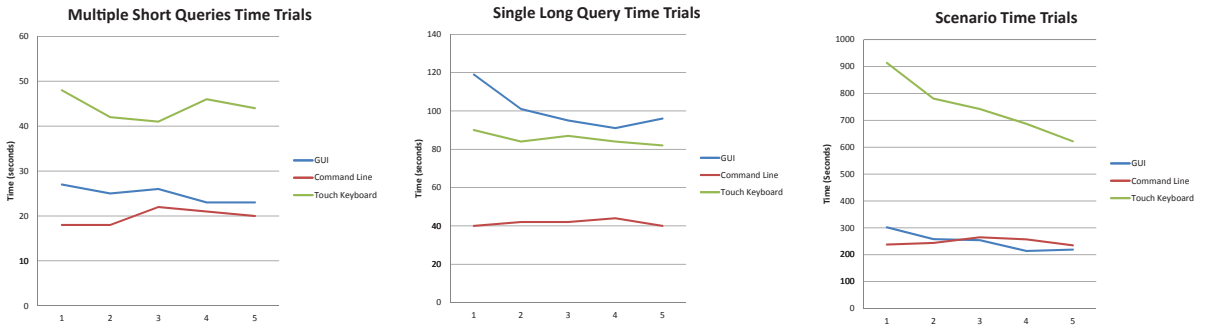


Fig. 4. Flow of the activities in the application

5. Experimental Evaluation

As a pilot study to evaluate the efficiency of the graphical query GUI, we compared it with two other forms of query entry; command line text entry on a full keyboard, and a touch screen keyboard on a 7" tablet. Three different types of time trials were executed; multiple short queries time trial, single long query time trial, and a scenario time trial. Each set of time trial had five trials.



(a) Multiple Short Queries (b) Long Query (c) Scenario

Fig. 5. Time Trial Graphs

From figure 5a, we can visually see that the graphical query and the command line entry time trials were quite close, with the touch screen keyboard entry having significantly longer times. The results of this set of time trials, distinctly show the advantage of using the graphical query over touch screen keyboards. The disadvantage that the graphical query had in these time trials was the fact that there was not enough opportunity to reuse large amounts of a query, and so the user had to create the query from a blank select node multiple times, and using command line to create a short query, even from a blank slate is faster for someone with 60-70 wpm typing speed.

From figure 5b, we can see that the GUI was actually slightly worse than the touch screen keyboard, once you get past the learning curve. The GUI had the same disadvantage in this set of time trials as it did

in the multiple short queries time trials, where the GUI had no opportunity to reuse a previous query, and had to add each node to the canvas before it was able to use it in a query which increases the time it takes to create a query.

Figure 5c, demonstrates how effective the GUI would be in typical use in comparison with command line entry and touch screen keyboard entry. Since the GUI allows a user to reuse and modify previous queries, it was just as efficient as command line entry on a full computer keyboard.

6. Conclusion and Future Work

We have presented and implemented a working Android application which allows users to utilize their mobile devices as a means to effectively and efficiently create complex queries. We have taken into account that Android devices have touch screen technology and created an interface that not only takes that into account, but allows a user to manipulate a visual representation of the query. The application was justified by its ability to increase the speed in which an individual can create a query on a touch screen.

Future work we have prototyped and plan to incorporate into the application include a force simulation means of organizing the nodes on the canvas dynamically. The force simulation includes the nodes being held together by the connections as if the connections were springs, and for the nodes to repel one another as a means to help distribute the nodes on the canvas. The type checking system would also apply an attraction force between the held node and the compatible nodes on the canvas, and a repulsion force between the held node and the incompatible nodes on the canvas.

References

[1] J. Rico, S. A. Brewster, Usable gestures for mobile interfaces: evaluating social acceptability, in: CHI, 2010, pp. 887–896.
[2] A. Bragdon, E. Nelson, Y. Li, K. Hinckley, Experimental analysis of touch-screen gesture designs in mobile environments, in: CHI, 2011, pp. 403–412.
[3] D. Wigdor, S. Williams, M. Cronin, R. Levy, K. White, M. Mazeev, H. Benko, Ripples: utilizing per-contact visualizations to improve user interaction with touch displays, in: UIST, 2009, pp. 3–12.
[4] Y. Li, Gesture search: a tool for fast mobile data access, in: UIST, 2010, pp. 87–96.
[5] H. Lü, Y. Li, Gesture avatar: a technique for operating mobile user interfaces using gestures, in: CHI, 2011, pp. 207–216.
[6] J. Ruiz, Y. Li, E. Lank, User-defined motion gestures for mobile interaction, in: CHI, 2011, pp. 197–206.
[7] M. V. Kleek, M. S. Bernstein, K. Panovich, G. G. Vargas, D. R. Karger, M. M. C. Schraefel, Note to self: examining personal information keeping in a lightweight note-taking tool, in: CHI, 2009, pp. 1477–1480.
[8] G. Oleksik, M. L. Wilson, C. S. Tashman, E. M. Rodrigues, G. Kazai, G. Smyth, N. Milic-Frayling, R. Jones, Lightweight tagging expands information and activity management practices, in: CHI, 2009, pp. 279–288.
[9] I. F. Cruz, A. O. Mendelzon, P. T. Wood, A graphical query language supporting recursion, in: Proceedings of SIGMOD'87, SIGMOD '87, 1987, pp. 323–330.
[10] A. Papantonakis, P. J. H. King, Gql, a declarative graphical query language based on the functional data model, in: Proceedings of the workshop on Advanced visual interfaces, AVI '94, 1994, pp. 113–122.
[11] D. Braga, A. Campi, S. Ceri, Xqbe (xquery by example): A visual interface to the standard xml query language, ACM Trans. Database Syst. 30 (2005) 398–443.