DBSnap++: Creating Data-driven Programs by Snapping Blocks

Yasin N. Silva
Arizona State University
ysilva@asu.edu

Anthony Nieuwenhuyse
Arizona State University
avannie1@asu.edu

Thomas G. Schenk
Arizona State University
tgschenk@asu.edu

Alaura Symons
Arizona State University
asymons1@asu.edu

ABSTRACT

A key development in Computer Science Education has been the introduction of block-based programming environments where programs are created by connecting blocks and the focus is on the program's logic instead of its syntax. Most of these environments support conventional (imperative) programming instructions. More recently, some systems have been proposed to enable the specification of database queries. While these two types of environments have been independently studied previously, there is significantly less work on the development and study of integrated environments that allow the creation of complete data-driven programs (real-world like programs that integrate conventional instructions and database queries). This paper introduces DBSnap++, a web-based environment that enables the specification of dynamic data-driven programs. DBSnap++ supports the specification of intuitive database query trees, a new type of list that dynamically gets its content executing a database query, and programs that generate different results when the underlying data changes. This paper presents DBSnap++'s design and implementation details, an array of programs to demonstrate how it can be effectively used as a learning tool, and a thorough comparison with previous environments. DBSnap++ is publicly available and aims to enable learners to fully understand and utilize the capabilities of data-driven programs.

CCS CONCEPTS

• **Social and professional topics → Computer science education**; *Information systems education*;

KEYWORDS

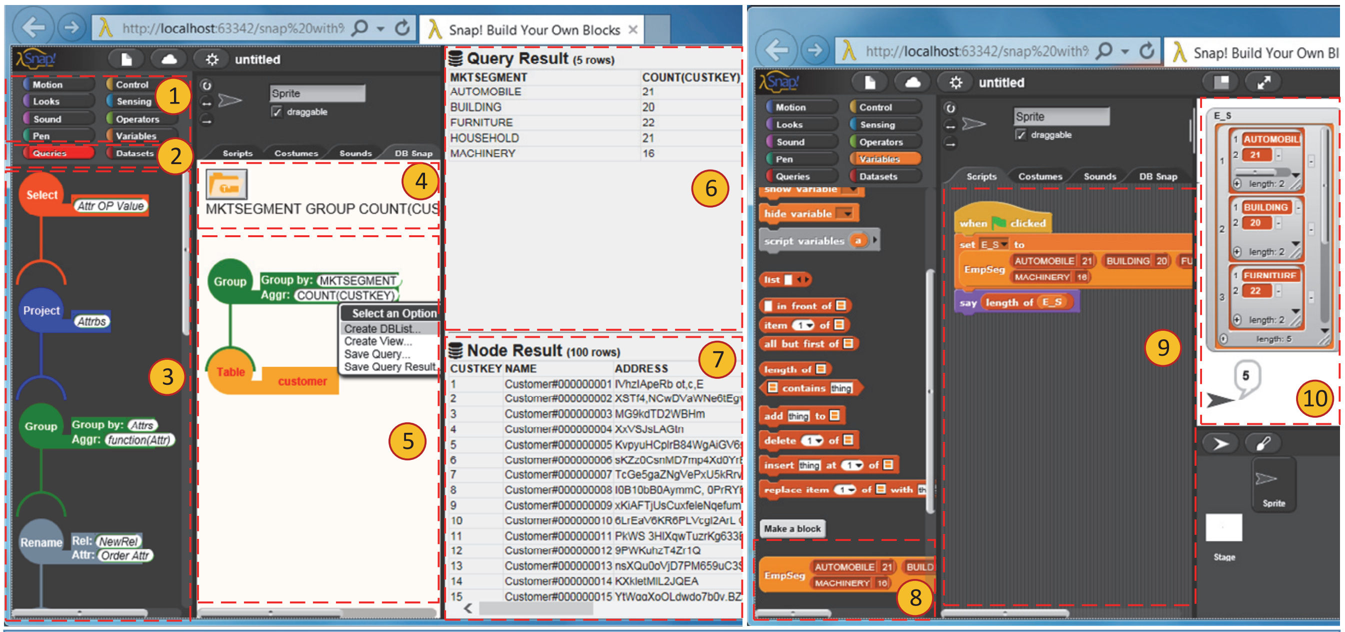
Data-driven programs, databases curricula; query languages

ACM Reference Format:
Yasin N. Silva, Anthony Nieuwenhuyse, Thomas G. Schenk, and Alaura Symons. 2018. DBSnap++: Creating Data-driven Programs by Snapping Blocks. In *Proceedings of 23rd Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE'18).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

A key challenge of learning programming languages is the need to create a set of instructions that is not only logically correct but also free of syntactical mistakes. The fact that even minor syntactical errors, like a missing semicolon, prevent the execution of logically correct programs is a source of frustration for many students. Block-based programming environments have effectively addressed this and other limitations of traditional approaches to learn computer programming. In these environments, instructions are represented by visual blocks and programs are built by snapping blocks together. The shape of a block provides additional guidance on how the block can be connected with others. Block-based environments have become popular tools and are transforming the way programming is taught, particularly to younger learners. Most of the previously proposed environments support conventional (imperative) programming instructions such as conditional and iterative constructs, e.g., Mindstorm [4], Scratch [5], Blockly [6], Snap! [9], and App Inventor [13]. Recently, some block-based systems have been proposed to enable the specification of database queries, e.g., DBSnap [10, 11] and Bags [2]. While these two types of environments have been independently studied, there is significantly less work on the development and study of environments that enable the integration of conventional and data-driven instructions. This paper introduces DBSnap++, a block-based environment to build data-driven programs that fully integrate conventional programming instructions with data-aware actions and blocks, e.g., database queries, lists that are dynamically linked to queries, and data manipulation actions such as data insertion and visualization. This paper presents the design and implementation details of DBSnap++ as well as sample programs that show how DBSnap++ is used to create data-driven programs like the ones needed in real-world scenarios. The key contributions of the paper are:

- The introduction of DBSnap++, a web-based and dynamic block-based environment that enables building programs that integrate conventional instructions and database queries.
- A detailed description of DBSnap++'s design and implementation guidelines. Our goal is to enable other researchers to extend or customize DBSnap++.
- The presentation of multiple data-driven programs that produce different results as the underlying data changes.
- A comparison with previously proposed block-based tools for conventional programming and database querying.
- The public availability of DBSnap++ [12] to be used by any instructor or student. DBSnap++ aims to have a transformational effect on data-driven programming learning.



Query Result (5 rows)

MKTSEGMENT	COUNT(CUSTKEY)
AUTOMOBILE	21
BUILDING	20
FURNITURE	22
HOUSEHOLD	21
MACHINERY	16

Node Result (100 rows)

CUSTKEY	NAME	ADDRESS
1	Customer#000000001	IVhzIApeRb ot,c,E
2	Customer#000000002	XSTf4,NCwDVaWNe6tEg
3	Customer#000000003	MG9kdTD2WBHm
4	Customer#000000004	XxVSJsLAGtn
5	Customer#000000005	KvpyuHCplrB84WgAiGV6
6	Customer#000000006	sKZz0CsnMD7mp4Xd0Yr
7	Customer#000000007	TcGe5gaZNgVePxU5kRr
8	Customer#000000008	I0B10bB0AymmC, 0PrRY
9	Customer#000000009	xKiAFTjUsCuxfeleNqefum
10	Customer#000000010	6LrEaV6KR6PLVcgl2ArL
11	Customer#000000011	PkWS 3HIXqwTuzrKg633
12	Customer#000000012	9PWKuhzT4Zr1Q
13	Customer#000000013	nsXQu0oVjD7PM659uC3
14	Customer#000000014	KXkletMIL2JQEA
15	Customer#000000015	YtWaaXoOLdwdo7b0v.BZ

User Interface Components: 1) standard block groups, 2) data-aware block groups, 3) block palette, 4) relational algebra panel, 5) query area, 6) query result panel, 7) node result panel, 8) DBLists, 9) script panel, 10) script result panel.

Figure 1: DBSnap++'s user interface.

The rest of the paper is organized as follows. Sections 2 and 3 present DBSnap++'s design and implementation details. Section 4 presents sample DBSnap++ data-driven programs. Section 5 presents a comparison of DBSnap++ with alternative tools. Section 6 presents a discussion of various facets of integrating DBSnap++ into the computing curricula and Section 7 concludes the paper.

2 DBSnap++'S DESIGN

The main visual components of DBSnap++, as shown in Fig. 1, are: conventional and data-aware blocks (items 1, 2 and 3), the query specification area (items 4 and 5), query result panels (items 6 and 7), DBLists (item 8), and the scripting area (items 9 and 10). DBSnap++ was designed integrating and extending Snap! [9] and DBSnap [10] components. While Snap! supports only conventional blocks and DBSnap only database query blocks, DBSnap++ supports both of them. Furthermore, DBSnap++ introduces new blocks and program creation sequences to enable the specification of dynamic data-driven programs. DBSnap++ includes also multiple features not available in Snap! or DBSnap such as the support of graph generation, database views, saving and loading queries from disk, data insertion and modification, and more flexible data grids. Queries in DBSnap++ are specified by building the tree representation of the relational algebra expression. Queries are built intuitively by connecting operator and dataset blocks in the query area. As the user builds the query, the query result and relational algebra panels are dynamically updated with the current query result and relational algebra expression, respectively. After a database query is specified, the user can create a DBList associated with the query. The DBList is a new type of block introduced in DBSnap++ that looks like a list but whose data is dynamically obtained by executing the associated query. DBLists can be combined with other conventional blocks in the script panel to create data-driven programs. DBLists, in fact, can be used anywhere where regular (static) lists are used. Additional details of each component are presented next.

2.1 Conventional and Data-aware Block Palettes

DBSnap++ enables the use of conventional and data-aware blocks. Conventional blocks are available under standard block groups like Control, Motion, Operators and Variables. Data-aware blocks are available under the Datasets and Queries groups. The Datasets block palette shows the available datasets (relations or tables). Each dataset is represented by a dataset block. This palette includes by default a small university database composed of several datasets, e.g., Students, Courses, Professors, etc. This database enables the creation of multiple queries of varying complexity with relatively small results that can be easily inspected. The Datasets block palette can also be customised by importing datasets from CSV-like files. This feature is particularly useful to prepare lessons, assignments and projets using specialized datasets. Like in DBSnap [10], a DBSnap++ dataset block has a central text area that shows its name, a left circular handle to connect it with a parent block or node, and a distinguishing orange color. As shown in the left screenshot of Fig. 1, dataset blocks are always leaf nodes. The Queries block palette shows the query operators available in DBSnap++. DBSnap++ supports many relational algebra operators including core operators (e.g., Selection, Projection and Rename), join operators (e.g., Theta-join and Natural-join), set operators (e.g., Cross Product, Union, Difference and Intersection), and highly useful extensions like the Aggregation/Grouping operator. Each operator type has a distinctive color. The structure of a DBSnap++ queries is very similar to the intuitive structure of query trees commonly used in database books. The shapes of dataset and query operator blocks facilitate the manipulation of blocks and the construction of queries. The shape of a query operator block has three main visual components: (1) top-left: a circular handle to connect the operator with its parent block, (2) top-right: a text area to specify required parameters, and (3) bottom: connection links to link this operator with its operand(s). The shape of operators facilitates detecting incomplete operators (with missing operands or parameters) and prevents the addition of more operands than needed.

2.2 Query Specification Area and Query Result Panels

Database query trees are constructed in DBSnap++'s query area. The use of trees to represent queries is a very useful analogy to learn about database queries (relational algebra and structured query language or SQL). A query tree clearly shows the datasets used in the query (leaf nodes) and the way these datasets are processed or combined by intermediate operator nodes. In fact, the query processing engine of many database systems transforms queries into query plan trees before optimizing and executing them. DBSnap++'s user interface includes a relational algebra panel that shows the corresponding relational algebra expression. The expression is dynamically updated after any change in the query.

As the user builds a query, the query results are dynamically shown in the query result panel. This panel is automatically updated after any changes in the query. This feature allows the user to quickly explore the effects of adding, customizing, or re-arranging operators. DBSnap++'s user interface also enables the exploration of individual query nodes. To do this, the user only needs to click on any intermediate query node and the node results will be loaded in the node result panel. Furthermore, when the user clicks on a dataset node, the node result panel becomes editable and the user can add, remove or modify the records of the dataset. Both results panels support also the visualization of their current content using various charts, e.g., area, bar, histogram, line and pie charts.

2.3 DBLists and Data-driven Programs

A key feature of DBSnap++ is the integration of database queries and conventional programming instructions in a single program. To achieve this, DBSnap++ introduces a new type of block named DBList (database connected list). A DBList block, shown in item 8 of Fig. 1, is a block that looks similar to a two-dimensional list block (list of lists) and has a distinctive orange color different than the color of regular lists. A DBList block is connected to a database query and dynamically retrieves its content by executing the query. This connection is dynamic, i.e., the DBList content will be updated anytime the associated query is modified or whenever the underlying data is updated. DBLists can be creating using the "Create DBList" context menu option of a query tree. This action will create a DBList block under the Variables block group (which also contains the standard list block). The way the DBList content is populated is by adding an internal list for every record of the query result. Since all the records have the same attributes, all the internal lists will have the same number of elements. DBSnap++ enables the modification of the query associated with a DBList even after the query tree has been deleted from the query area. This can be done by selecting the context menu option to edit the query of a DBList. This tool will open a new query specification area, load the current query, and process any query changes.

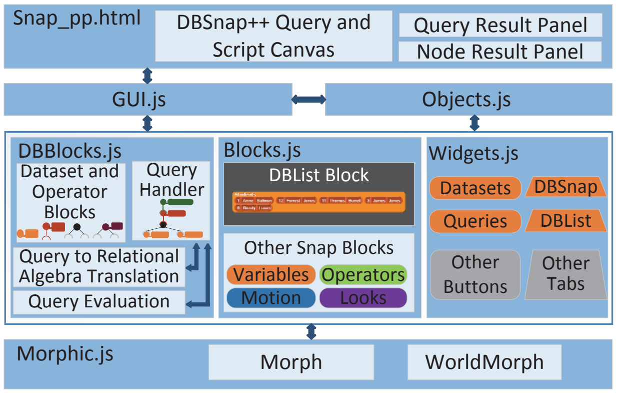


Figure 2: DBSnap++ architecture.

After a DBList block has been created, it can be used in any location where a regular list block could be used. As it is also the case in Snap!, regular scripts, which in the case of DBSnap++ can now include DBLists, are created using the script and script result panels. Moreover, DBSnap++ supports the use of multiple instances of a DBList and is aware of the connection between each instance and the corresponding query. When the query associated with a DBList is updated, the changes are propagated to all its instances, i.e., the content and visual representation of all the instances are properly updated.

3 IMPLEMENTING DBSnap++

DBSnap++ was implemented as a web application that uses standard internet browser features (HTML5 and JavaScript). The use of these technologies enables running DBSnap++ under many internet browsers and hardware devices, e.g., desktop computers, laptops and tablets. Fig. 2 shows the main components of DBSnap++'s architecture: an HTML web page (Snap_pp.html) and six core JavaScript libraries (GUI.js, Objects.js, DBBlocks.js, Blocks.js, Widgets.js and Morphic.js). DBSnap++ was built integrating and extending modules from Snap! (Snap.html, GUI.js, Objects.js, Blocks.js, Widgets.js) [9] and DBSnap (DBSnap.html, GUI.js, DBBlocks.js) [10].

Snap_pp.html has three container areas: the DBSnap++ Query and Script Canvas (composed of the block palettes, and the query and scripting areas), and the query and node result panels. Snap_pp.html interacts with GUI.js to set and adjust the position of the HTML containers, support the manipulation of blocks, and display the results of queries and scripts. GUI.js defines a global container structure that stores general properties. Objects.js defines smaller containers that can hold blocks and graphical objects. In DBSnap++, GUI.js and Objects.js were extended to enable the use

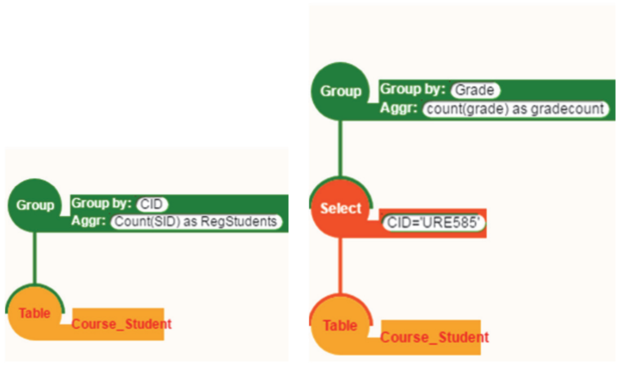
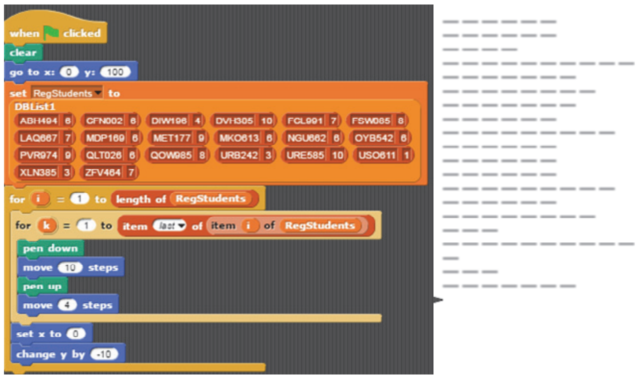


Figure 3: Queries used in the sample programs.



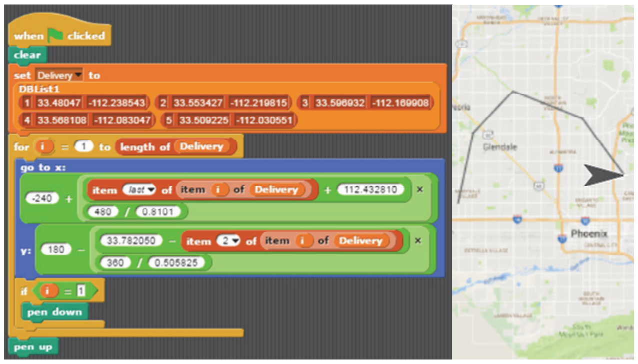
```
when clicked
clear
go to x: 0 y: 100
set RegStudents to
  DBList1
  ABH494 6  CFN002 6  DIW196 4  DVH305 10  FCL991 7  FSW085 8
  LAQ667 7  MDP169 6  MET177 9  MKO613 6  NGU662 6  OYB542 6
  PVR974 9  QLT026 6  QOW985 8  URB242 3  URE585 10  USO611 1
  XLN385 3  ZFV464 7
for i = 1 to length of RegStudents
  for k = 1 to item last of item i of RegStudents
    pen down
    move 10 steps
    pen up
    move 4 steps
  set x to 0
  change y by -10
```

Figure 4: Program that generates a bar graph.

of query blocks and DBLists. GUI.js and Objects.js interact with DBBlocks.js, Blocks.js and Widgets.js to support the operation of query and conventional blocks. DBBlocks.js controls the visual properties and behavior of que query operator and dataset blocks. The Query Handler module of DBBlocks.js keeps track of the internal structure of a query and interacts with the Query Evaluation and Query to Relational Algebra Translation modules to generate the query result and relational algebra expression, respectively. Blocks.js contains the specification of conventional scripting blocks and was extended with the addition of DBLists, special blocks that look similar to lists but are dynamically linked to database queries. Widgets.js is a library that specifies the actions associated with buttons and tabs. This module was extended to properly support mouse events on the tabs and buttons used to create and edit queries and DBLists. Particularly, this module propagates the changes made on the query of a DBList to all the linked DBList instances. Morphic.js, an open-source library developed by Jens Mönig, provides basic classes such as Morph and WorldMorph which specify the core behavior of blocks and define a basic canvas where blocks can be manipulated, respectively.

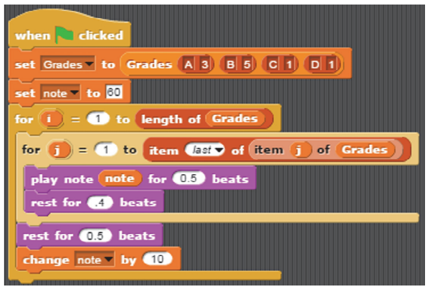
4 DATA-DRIVEN PROGRAMS

DBSnap++ enables users to create data-driven programs in a wide range of application scenarios. These programs are similar to real-



```
when clicked
clear
set Delivery to
  DBList1
  1 33.48047 -112.238543  2 33.553427 -112.219815  3 33.596932 -112.169908
  4 33.568108 -112.083047  5 33.509225 -112.030551
for i = 1 to length of Delivery
  go to x: -240 + ((item last of item i of Delivery) + 112.432810) × (480 / 0.8101)
        y: 180 − (33.782050 − item 2 of item i of Delivery) × (360 / 0.505825)
  if i = 1
    pen down
pen up
```

Figure 5: Program that visualizes a delivery schedule.



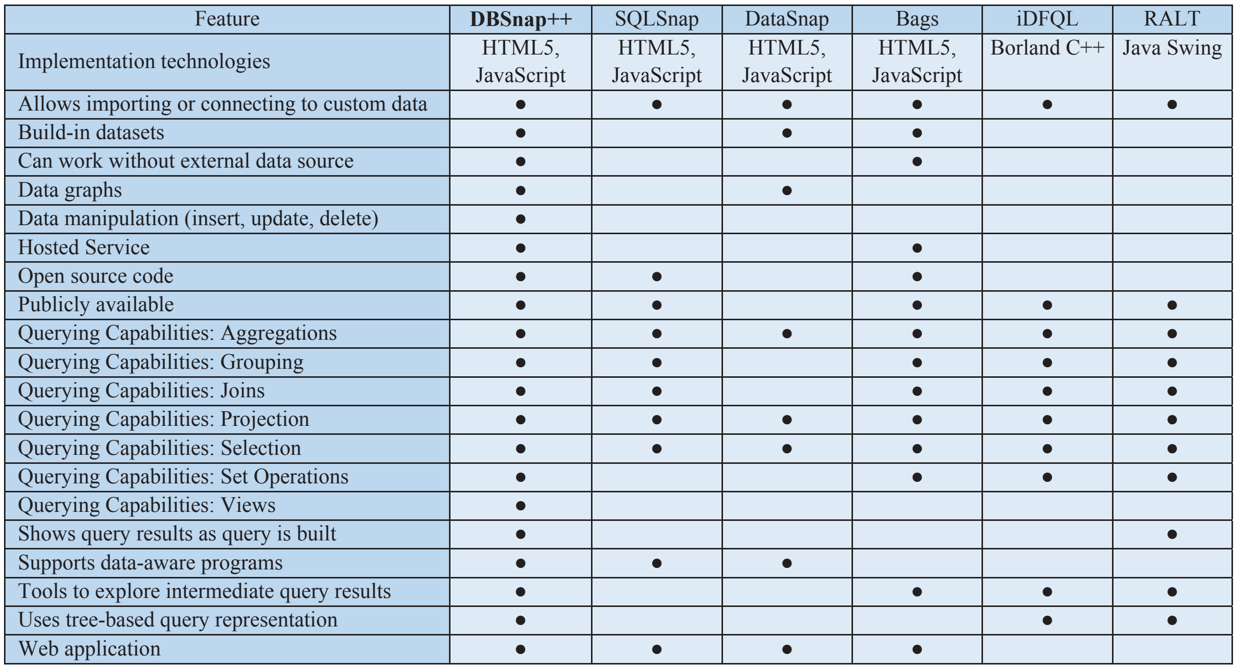
```
when clicked
set Grades to Grades A 3 B 5 C 1 D 1
set note to 60
for i = 1 to length of Grades
  for j = 1 to item last of item j of Grades
    play note note for 0.5 beats
    rest for .4 beats
  rest for 0.5 beats
  change note by 10
```

Figure 6: Program that represents data using sounds.

word programs that retrieve data using database queries and further process, analyze or visualize it. Since DBSnap++ programs do not have to rely on static data, they are able to produce different results as the underlying data or queries change. A key property of DBSnap++ programs is that they integrate imperative instructions and queries. This feature differentiates DBSnap++ from tools that only support queries (e.g., Bags [2] and DBSnap [10]) or imperative programs (e.g., Scratch [5] and Snap! [9]). This section presents several sample DBSnap++ programs. Due to space constrains, each program contains only the core components to showcase a useful application. These programs can be easily extended to incorporate additional features.

4.1 Generating Bar Diagrams

DBSnap++ can be used to create visual representations of query results. This example uses a query to count the number of students registered in each university course and displays this information using a bar chart. The program uses the Course_Student table of DBSnap++'s University database, which contains two attributes: CID (course ID) and SID (student ID). The query, shown in the left image of Fig. 3, includes a grouping operator to generate an aggregated record per course. The program, shown in Fig. 4, cleans the stage, resets the pointer, and uses a DBList block linked to the previously described query. After this, the program includes two nested loops to create the bar chart. The outer loop prints a bar for each course, and the inner loop prints the individual line segments of each bar. Fig. 4 also shows a sample of the output.



Feature	**DBSnap++**	SQLSnap	DataSnap	Bags	iDFQL	RALT
Implementation technologies	HTML5, JavaScript	HTML5, JavaScript	HTML5, JavaScript	HTML5, JavaScript	Borland C++	Java Swing
Allows importing or connecting to custom data	●	●	●	●	●	●
Build-in datasets	●		●	●		
Can work without external data source	●			●		
Data graphs	●		●			
Data manipulation (insert, update, delete)	●					
Hosted Service	●			●		
Open source code	●	●		●		
Publicly available	●	●		●	●	●
Querying Capabilities: Aggregations	●	●	●	●	●	●
Querying Capabilities: Grouping	●	●		●	●	●
Querying Capabilities: Joins	●	●		●	●	●
Querying Capabilities: Projection	●	●	●	●	●	●
Querying Capabilities: Selection	●	●	●	●	●	●
Querying Capabilities: Set Operations	●			●	●	●
Querying Capabilities: Views	●					
Shows query results as query is built	●					●
Supports data-aware programs	●	●	●			
Tools to explore intermediate query results	●			●	●	●
Uses tree-based query representation	●				●	●
Web application	●	●	●	●		

Figure 7: Comparison of block-based learning tools for database querying and programming.

4.2 Using Maps and Geographic Coordinates

DBSnap++ can also be used to process and represent geographical data. In this example, DBSnap++'s import features are used to load a dataset with the list of a delivery schedule and a map of the corresponding delivery area. This example uses table DeliverySchedule with attributes: Date, ClientID, Latitude, Longitude, and OrderID. The query has a projection operator that retrieves the latitude and longitude coordinates. The program, shown in the left section of Fig. 5, (1) has an equation to scale the coordinates based on the size of the map window (480x360) using the top left corner as a starting point, and (2) moves to each point (x, y) without lifting the pen to draw the path of the delivery schedule. The output of the program is presented in the right section of Fig. 5.

4.3 The Sound of Data

DBSnap++ can also be used to represent data for the visually impaired using sounds instead of visual artifacts. This example uses different notes to represent different data values. The query for this example also uses the Course_Student table of DBSnap++'s University database. As shown in the right image of Fig. 3, the query has a select block to retrieve the records of a given course, and a grouping operator to compute the number of students that received each grade. The program, shown in Fig. 6, specifies a DB-List block linked to the previously described query and includes two nested loops to create the sounds. Each cycle of the outer loop plays the beeps associated with a given letter grade (one beep per student receiving that grade). The note to be used in the beeps is increased at the end of each cycle. This ensures that each grade has a unique identifying sound. Each cycle of the inner loop produces a beep sound at the current note.

5 RELATED LEARNING TOOLS

This section presents and compares several block-based tools that have been previously proposed for database querying and programming. Fig. 7 summarizes the features of all the compared tools. The remaining part of this section describes the core properties of each alternative tool and compares it with DBSnap++.

SQLSnap. SQLSnap [8] is a Snap! [9] extension that allows the user to build SQL queries to be executed on a MySQL database. A key difference between SQLSnap and DBSnap++ is that SQLSnap does not use the tree-based query representation. SQLSnap's query block closely mirrors the potentially complex text version of SQL queries. SQLSnap works with a connection to an external MySQL database or can read data from a text file, although this is limited to one text file at a time. While SQLSnap is a web-based application, it must be installed on an appropriate server (e.g. LAMP, XAMPP) with MySQL and PHP installed in order to interact with a database. DBSnap++ does not need installation, has datasets preloaded, and multiple additional datasets can be easily uploaded from text files.

DataSnap. DataSnap [3] is a Snap! [9] extension that uses a RESTful API to import real-time data (cloud-based and CSV files). This data can then be processed within Snap!. DataSnap integrates also some blocks to visualize data on maps. DataSnap supports a SQL-like language but has limited querying capabilities, e.g., does not support joins, grouping and set operators. Furthermore, DataSnap's queries closely follow the text version of SQL queries and do not use the query tree representation. Even though the manuscript presenting this tool [3] states that DataSnap is open source, we could not find the source code in the web.

Bags. Bags [2] is a Snap! modification that supports the specification of relational algebra queries by manipulating and connecting blocks. Like DBSnap++, Bags allows the visualization of the query results and includes several built-in datasets. Two key differences between Bags and DBSnap++, however, are related to query representation and the type of supported programs. In Bags, queries are represented using blocks that look like the standard Snap! blocks. DBSnap++ uses customized block shapes and represents queries using an intuitive tree structure. Also, while DBSnap++ supports specifying complete data-driven programs, Bags only supports the construction of queries which cannot be integrated with conventional programming blocks. Bags allows the user to explore the results of individual query blocks but does not automatically update the query results as the user constructs the query. DBSnap++ updates the node and query results in real-time.

iDFQL. The iDFQL (Interactive Data Flow Language) tool [1] is a graphical system that allows the creation of relational algebra queries. Queries in this tool take the form of flow diagrams composed of specialized blocks and links. iDFQL requires a connection with an external database. A fundamental difference between iDFQL and DBSnap++ is that, unlike DBSnap++, iDFQL does not support conventional programming blocks and consequently does not support the creation of data-driven programs that use the specified queries. Moreover, connecting blocks in iDFQL is not guided by the shape of the blocks. Instead, connection links need to be added separately. DBSnap++ blocks have connection handlers and connection links that intuitively guide the query construction process. Similarly, specifying operator predicates in iDFQL requires the addition of separate blocks while in DBSnap++ they are integral components of the operator blocks. iDFQL has compatibility issues with Windows 7 and later and is not actively maintained.

RALT. The RALT (Relational Algebra Learning Tool) application [7] allows the creation of relational algebra queries but, unlike DBSnap++, does not allow the construction of data-driven programs using these queries. While RALT and DBSnap++ use trees to represent queries, RALT uses an extended tree structure that requires the addition of intermediate nodes to (1) visualize the results of intermediate query operators and (2) specify operator predicates. The resulting query trees can quickly grow and fill the limited query construction area. Furthermore, unlike DBSnap++, RALT requires an external database connection and does not generate the associated relation algebra expression of the constructed query.

6 DISCUSSION

While DBSnap++ can be used as a learning tool to teach imperative programming and database querying separately, it is particularly useful for teaching data-driven programming (the creation of programs that process data retrieved from databases). This subject can be covered as part of programming or database courses. An important related learning objective in this area is that students are able to create programs that effectively retrieve data from a database using a query language and programmatically process the query results. Here, DBSnap++ can be used as an integrated environment that enables students to build programs that combine database queries of varying complexity and conventional programming instructions to process their results. The query results can be used for example to (1) specify program units with data-dependent actions, (2) create visualizations, and (3) further analyze the retrieved data. Examples of these applications are provided in Sec. 4. Moreover, DBSnap++ can be used to show how data changes over time and how these changes affect the behavior of the programs students build. The robust set of DBSnap++'s querying features makes it also an effective tool to teach database querying in database courses. Some of these features, such as the support of data insertion, modification and deletion, graph generation, and database views, are not currently available in DBSnap. In this context, students can use DBSnap++ to interactively explore each query operator, build queries that combine multiple operators, and learn the relationships between query trees and their relational algebra expressions. Since DBSnap++ can be used with custom datasets and is a publicly hosted web application [12], instructors can use it in a wide range of class activities and hardware devices.

7 CONCLUSIONS

Having a clear understanding of database query languages and being able to build programs that use the query results for various processing and visualization tasks are key requirements of many computing job positions. This paper introduces DBSnap++, a block-based tool aimed at facilitating the learning of data-driven programming that integrates database queries and conventional programming instructions. To the best of our knowledge, DBSnap++ is the only tool that enables building data-driven programs where queries are specified using an intuitive tree-based structure similar to the one used by many textbooks and educators. DBSnap++ is also an interactive tool that automatically shows any effects of data or query modifications in the program's output. This paper presents the design and implementation details of DBSnap++, shows several programs enabled by the tool, and presents a detailed comparison with other tools.

REFERENCES

[1] Ana P. Appel, Elaine Q. Silva, Caetano Traina, and Agma J. M. Traina. 2004. iDFQL: A Query-based Tool to Help the Teaching Process of the Relational Algebra. In *WCETE*.

[2] Jason Gorman, Sebastian Gsell, and Chris Mayfield. 2014. Learning Relational Algebra by Snapping Blocks. In *ACM SIGCSE*.

[3] Jonathon D. Hellmann. 2015. *DataSnap: Enabling Domain Experts and Introductory Programmers to Process Big Data in a Block-Based Programming Language.* Master's thesis. Virginia Tech, Virginia, USA.

[4] Seung H. Kim and Jae W. Jeon. 2007. Programming LEGO mindstorms NXT with visual programming. In *ICCAS*.

[5] John H. Maloney, Kylie Peppler, Yasmin Kafai, Mitchel Resnick, and Natalie Rusk. 2008. Programming by Choice: Urban Youth Learning Programming with Scratch. In *ACM SIGCSE*.

[6] Assaf Marron, Gera Weiss, and Guy Wiener. 2012. A Decentralized Approach for Programming Interactive Applications with JavaScript and Blockly. In *AGERE!*

[7] Pritam Mitra. 2009. *Relational Algebra Learning Tool.* Technical Report. Dept. of Computing, Imperial College.

[8] Eckart Modrow. 2014. SQLsnap! http://snapextensions.uni-goettingen.de. (2014).

[9] Chris North and Ben Shneiderman. 2000. Snap-together Visualization: Can Users Construct and Operate Coordinated Visualizations? *Int. J. Hum.-Comput. Stud.* 53, 5 (2000), 715–739.

[10] Yasin N. Silva and Jaime Chon. 2015. DBSnap: Learning Database Queries by Snapping Blocks. In *ACM SIGCSE*.

[11] Yasin N. Silva and Jaime Chon. 2015. Querying databases by snapping blocks. In *IEEE ICDE*.

[12] Yasin N. Silva, Anthony Nieuwenhuyse, Thomas Schenk, and Alaura Symons. 2018. DBSnap++. http://www.public.asu.edu/~ynsilva/dbsnapplus. (2018).

[13] David Wolber. 2011. App Inventor and Real-world Motivation. In *ACM SIGCSE*.