On Teaching Big Data Query Languages

Thanaa Ghanem
Information and Computer Science Department
Metropolitan State University
St. Paul, MN, 55106
thanaa.ghanem@metrostate.edu

Abstract

Big data computing systems (e.g., Hadoop) have recently seen tremendous intake as computing platforms for data-intensive applications. The emergence of such big data computing systems has triggered a plenty of new techniques for data management. For example, several new query paradigms have been introduced including *map-reduce*, *HiveQL*, *Impala*, *Pig Latin*, and *Spark*. In order to cope with this big-data surge and hence meet the current job market requirements, computer science students need to have a good understanding of the big-data technologies. In this paper, we give a module to teach the basics of three big-data query languages. First, we give a categorization of big-data languages into three categories: procedural, declarative, and scripting. Then we pick one language from each category and show how a given query can be expressed by the three syntactically-different languages. We mainly focus on queries that are composed of one or more of the various relational algebra operators (e.g., select, project, join, and group by). We believe that the process of contrasting the languages helps students to gain deeper understanding of the expressive power similarities between the various languages, and hence it will be easier to learn new languages as they are being introduced. Throughout this paper we will give various teaching resources that can be used by instructors to teach the proposed module.

1 Introduction

Large volumes of data are being generated and collected everyday from sources like sensor networks and social media networks. Businesses consider these data sets as invaluable resources that can be used to extract information and insights to promote their businesses. As a result, and to accommodate to this data surge, big data computing systems such as Hadoop [16] have seen tremendous intake in recent time as platforms for the management of big-data sets with high volume, velocity, and variety. The emergence of such big data computing systems has triggered a plenty of new techniques for data storage, management, analysis, and visualization.

In order to cope with the big-data surge and hence meet the current job market requirements, the undergraduate Computer Science curriculum needs to be enriched with modules to equip students with the needed knowledge and skills. The following are three different approaches that can be taken by Universities to address the big-data teaching requirements: (1) introduce new big-data degrees (e.g., majors, minors, or certificates), (2) introduce big-data courses as electives, or (3) incorporate big-data teaching modules in existing courses. The third approach can be immediately applied by any computer science department as it requires minimum additional resources. Big-data teaching modules can be incorporated in several fundamental Computer Science courses including *Operating Systems*, *Computer Organization and Architecture*, and *Database Management Systems*.

In this paper, we introduce a module to introduce the basics of three big-data languages as a part of a database course. A classical database course covers the following topics: the relational data model, relational algebra, and the Structured Query Language (SQL). Our module then focuses on teaching how a given SQL query can be expressed using three big-data query paradigms, namely *map-reduce* [17], *Pig Latin* [8], and *Impala* [6]. The proposed module can also be extended and taught as an introductory course on big-data.

Previous works introduced modules to teach each of the query paradigms independently. For example, [10] introduces a module to teach map reduce in an introductory computer science course, [9] introduces teaching map reduce in a distributed systems course. On the other hand, [13] introduces a module to teach several SQL implementations in various big data systems including MongoDB and HBase. Moreover, [14] introduces modules to incorporate big data in the computing curriculum through separate map-reduce and SQL modules. Although in a production environment, a certain language may be chosen based on the complexity of the underlying data sets and operations, however, from the pedagogical perspective, it is essential for computer science students to understand how syntactically-different languages are used to express semantically-similar queries for the following reasons: (1) this knowledge will enable students to learn new languages as they are introduced, and (2) by understanding the relationship between languages, students may contribute in the introduction of new languages or adding features to existing languages.

The rest of the paper is organized as follows. Sections 2 and 3 discuss the learning units included in our proposed module. Section 4 discusses how to set up a programming envi-

ronment that can be used for hands-on practice of the proposed topics. Finally, Section 5 concludes the paper.

2 Big-Data Query Languages Learning Units

As the number of big-data technologies increases, one of the first challenges for computer science educators is to categorize these different technologies into well-defined categories. Students then are taught the fundamental concepts of each category along with the similarities and differences among the various categories. This approach allow students to teach themselves new technologies as they are introduced in the future. In this paper, we introduce a categorization for big-data languages into three categories: procedural, declarative, and scripting. We first discuss the characteristics of each language category, then we pick an example language from each category to discuss in more details through examples. Mainly, we introduce the following three learning units to teach big-data query languages:

- Fundamentals of relational algebra.
- Categories of big-data query languages.
- Relational algebra operators in big-data languages.

The proposed learning units are discussed in Sections 2.1, 2.2, and 2.3 respectively. We illustrate our ideas using examples on a synthetic data set of meteorological station data [13]. The data set has two tables: *Stations(stationID, zipcode,latitude,longitude)* and *Reports (stationID, temperature, humidity, year,month)*. The data generator and a sample data set are available in [2].

2.1 Fundamentals of Relational Algebra

Although not all big-data query paradigms assume that data is stored in tables, however, almost all big-data query paradigms provide support for relational algebra operations because relational algebra is proved to be complete and very efficient in expressing data management operations. Hence, relational algebra operators provide good means to illustrate how syntactically-different query languages can be semantically similar. In this section, we summarize the fundamental relational algebra concepts that are used in the rest of the paper. More detailed explanations of relational algebra concepts can be found in any database text book, for example [12].

The ***Relational Data Model*** and ***Relational Algebra*** are the main power behind Relational Database Management (RDBSM) systems [3] . In the relational model, the term *relation* is used to refer to a table, the term *tuple* is used to refer to a row, and he term *attribute* refers to a column of a table. Basically, a relational database consists of a collection of tables, each of which is assigned a unique name. Relational algebra defines a set of operations that are used to express queries over database tables. In this paper, we will focus only on the following relational operators: *select* (σ), *project* (π) , *join*($\bowtie$), and *group by*(γ). Each operator is defined to take one or more relations as input and produce a relation as output. The

middle column in Table 2 gives the semantics of the various relational operators.Typically, a query in relational algebra is composed of sequence of relational operators where the output of one operator is the input for the following operator in the sequence.

Structured Query Language (SQL) is the most widely used implementation of relational algebra. The typical SQL query consists of three clauses: *select*, *from*, and *where*. Optional SQL constructs (e.g. *group by* and *aggregates*) are used to express a wider range of queries over relational tables. It is important to emphasize the close connection between SQL and relational algebra. Basically, each SQL query can be expressed as a relational algebra expression and vise versa. For example, consider the following query, Q_1 over our example database: Q_1*: Find the list of temperatures reported by stations in zipcode 55112.* Q_1 is represented in SQL as follows:

$$\begin{array}{l}\texttt{select } S.stationID,\ temperature\\ \texttt{from } Stations\ S,\ Reports\ R\\ \texttt{where } S.stationID\ =\ R.stationID\ and\ zipcode = 55112\end{array}$$

Q_1's corresponding relational algebra expression is as follows:

$$\pi_{Stations.stationID,temperature}(\sigma_{zipcod=55112}(Stations \bowtie_{Stations.stationID=Reports.stationID} Reports))$$

Query execution refers to the range of activities involved in extracting data from a database. The activities include translation of SQL queries into an internal form that can be used at the physical level for the actual evaluation of the query.This translation process is similar to the work performed by the parser of a compiler.The translation step mainly constructs relational-algebra expression which is then translated into a parse-tree of relational algebra operators. Each relational algebra operation is then executed by one of several different algorithms where the output of one operator is used as input to the next operator in the parse-tree. The sequence of primitive operations that can be used to evaluate a query is called *query-execution plan*. It is important to note that query optimization techniques can be used to transform a given query-execution plan to another equivalent plan that is more efficient, however, query optimization is beyond the scope of this paper.

The transformation of a given SQL query into the corresponding query-execution-plan is achieved as follows. First, a different leaf is created for each table listed in the *from* clause. Second, the root node is created as a project operation (π) involving the required output attributes listed in the *select* clause. Third, the qualifications listed in the *where* clause is translated into the appropriate sequence of *join* ($\bowtie$) or *select* (σ) operations going from the leaves to the root. Finally, if needed, a *group by* (γ) operator is added. For example, the query-execution-plan that corresponds to query Q_1 is shown in Figure 1.

2.2 Categories of Big-Data Query Languages

Teaching computer science students the similarities and differences among programming languages is more valuable than teaching them one or two specific programming languages for the following potential benefits: (1) increasing students' capacity to express ideas, (2) increasing students' ability

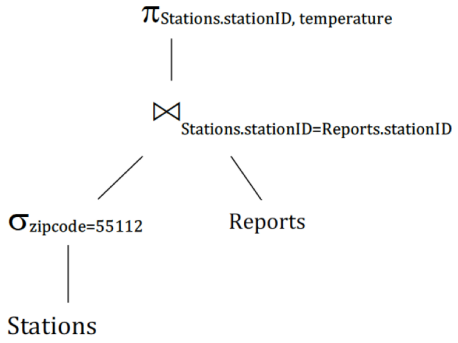

Figure 1: Query Execution plan for Q_1.

to learn new languages, and (3) overall advancement of computing [11]. The study of programming languages, like the study of natural languages, can be divided into examinations of *syntax* and *semantics*. The *syntax* is the language's expressions, statements, and program units. The language *semantics* is the meaning of those expressions, statements, and program units.

In the big-data era, a plethora of language are introduced to meet the growing need for ad-hoc analysis of extremely large data sets. We categorize big-data query languages into three categories: procedural, declarative, and scripting. In this section we discuss the characteristics of the language categories. Table 1 gives example languages in each category. It is important to note that all big-data programming languages are implemented to work on top of *Hadoop* and the underlying *Hadoop Distributed File System (HDFS)* and *the map-reduce programming paradigm* [16].

Category	Language
Procedural	Map-reduce in Java
Declarative	HiveQL and Impala
Scripting	Pig Latin and Spark

Table 1: Categories of Big-Data Query Languages.

2.2.1 Procedural Languages

A *procedural languages* is a language that describes a step-by-step algorithm to complete a computational task or program. In the big-data era, no new procedural languages have been introduced, however, the map-reduce framework [4] is introduced as a programming model and an associated implementation for processing large data sets with a parallel, distributed algorithm on a cluster.Existing programming languages (e.g., Java and C++) can be used to implement programs using the map-reduce framework concepts. The map-reduce framework mainly works by dividing the processing task into two main phases: *map* and *reduce*. The framework developer is required to provide the implementation of the *map* and *reduce* functions in a high-level programming language of choice (e.g., Java). Key-value pairs *(K,v)* form the basic data structure in map-reduce algorithms. Basically, the design of map-reduce algorithms involves imposing the key-value structure on the

input and output data sets. The *map* and *reduce* functions have the following signatures:

$$map : (k1, v1) \rightarrow list(k2, v2)$$
$$reduce : (k2, list(v2)) \rightarrow list(k3, v3)$$

The input to a map-reduce job starts as data stored on the underlying distributed file system. The mapper is applied to every input key-value pair to generate an arbitrary number of intermediate key-value pairs. The reducer is applied to all values associated with the same intermediate key to generate output key-value pairs. Implicit between the map and reduce phases is a distributed group by operation on intermediate keys. More detailed explanation of the map-reduce frameworks can be found in [7].

2.2.2 Declarative Languages

A declarative language is a high-level language in which the programmer specifies what needs to be done rather than how to do it. Structured Query Languages (SQL) is one of the most commonly used declarative languages that is used in relational DBMS. HiveQL [15] and Impala [6] are two declarative big-data languages that use a syntax that is very similar to SQL. The compilers of HiveQL and Impala transforms the input SQL queries into the corresponding map-reduce jobs that are executed on the Hadoop cluster, hence, saving the developers the cost of writing custom map-reduce programs.

2.2.3 Scripting Languages

A scripting language is language that employs a high-level construct to interpret and execute one command at a time. Pig Latin [8] is a big-data scripting language that is designed to fit in a sweet spot between the declarative style of SQL, and the low-level, procedural style of map-reduce. A Pig Latin script is a sequence of steps each of which carries out a single data transformation. At the same time, the transformations carried out in each step are fairly high-level, e.g., filtering, grouping, and aggregation, much like in SQL. For example, Q_1 is expressed in Pig Latin in three steps as follows:

$selectedStations =$ **filter** $Stations$ **by** $zipcod = 55112$
$joinOutput =$ **join** $selectedStations$ **by** $stationID,\ Reports$ **by** $stationID$
$queryOutput =$ **foreach** $joinOutput$
generate $selectedStations.stationID,\ Reports.temperature$

In effect, writing a Pig Latin program is similar to specifying a query execution plan. For example, the previous three statements corresponds to performing select(σ), join($\bowtie$), then project(π) in order.

2.3 Relational Algebra in Big-Data Languages

Relational algebra operators provide good means for teaching the basic syntax and semantics of the various big-data query languages. In this section, we contrast how relational algebra is supported by the three big-data language categories. For demonstration purposes, we will use the general map-reduce algorithm as an example procedural language, Pig Latin as an example scripting language, and SQL as an example declarative language. After studying each operator independently, we will show how to express a query that is composed of multiple relational operators in Section 3. The fourth and fifth columns of Table 2 give how each relational operator is supported in *map-reduce*

Operator	Algebric symbol	Semantics	Map-reduce	Pig Latin
select	$\sigma_C(R)$	Given a relation R, $\forall t_R \in R$, if t_R qualifies the condition C, produce t_R in the output	***map:*** $\forall t_R \in R$, if t_R qualifies C, produce (t_R, t_R) in the output. ***reduce:*** the reduce function is the identity reducer that simply passes each key-value pair to the output.	***filter*** R ***by*** C
project	$\pi_{a_R}(R)$	Given a relation R and a subset a_R of the attributes of R, $\forall t_R \in R$ produce only a_R attributes in output	***map:*** $\forall t_R \in R$, generate a tuple t'_R with only a_R attributes and produce $(t'_R, 1)$ in output. ***reduce:*** the reduce function is the identity reducer that simply passes each key-value pair to the output.	***foreach*** R ***generate*** a_R
join	$R \bowtie_{R.a_{R_1}=S.a_{S_1}} S$	Given two input relations, R and S, compare each pair of tuples, one from each relation. If $R.a_{R_1} = S.a_{S_1}$, then produce in the output a tuple that has all attributes from both R and S	**map:** $\forall(a_{R_1}, a_{R_2}) \in R$, produce $(a_{R_1}, (1, a_{R_2}))$ in output and $\forall(a_{S_1}, a_{S_2}) \in S$, produce $(a_{S_1}, (2, a_{S_2}))$ in output. **reduce:** $\forall$ Key value a_{R_1}: for every pair of $(1, a_{R_2})$ tuple and $(2, a_{S_2})$, produce in the output a tuple of the form $(a_{R_1}, (a_{R_2}, a_{S_2}))$.	**join** R **by** a_{R_1}, S **by** a_{s_1}
group by and Aggregation	$\gamma_{a_{R_1}, \theta(a_{R_2})}(R)$	Given a relation R, partition its tuples into groups where all tuples with the same values of a_{R_1} are placed in the same group. Then, for each group, aggregate the values using the aggregate function $\theta(a_{R_2})$.	**map:** $\forall t_R \in R$, produce in the output a tuple (a_{R_1}, a_{R_2}). **reduce:** Let x = the result of applying θ to the list of a_{R_2} values associated with key a_{R_1}. produce (a_{R_1}, x) in the output.	temp = **group** R **by** a_{R_1} output = **foreach** temp **generate** group, $\theta(a_{R_2})$

Table 2: Relational Algebra Operators in Big-Data Languages

and *Pig Latin* languages respectively. Impala is not shown in the table because it uses the same syntax as SQL. More detailed explanation about supporting relational algebra operators in map-reduce can be found in [5]

The meanings of the symbols in Table 2 are as follows: R and S are table names, t_R and t_S refer to tuples from tables R and S respectively, a_R and a_S refers to subsets of attributes from tables R and S respectively.

3 Syntactically-different Representation of the Same Query

In this section, we give examples on how to represent a given relational algebra query using four different formats, namely, query semantics, SQL, map-reduce, and Pig Latin. The goal of these examples is to provide students with a deep understanding of the correspondence of the four query representations. An instructor can prepare several other exercises using different datasets as hands-on in-class exercises to further practice these concepts. Exercises can be in different formats, for

example, one exercise can be given a Pig Latin script, ask students to write the corresponding SQL query. Another exercise is to give students a query expressed in SQL and ask students express the query semantics in one or two English sentences. As shown in Figure 2, students should practice how to perform different mappings between the four query representations (e.g., from SQL to semantics, semantics to SQL, SQL to Pig Latin, Pig Latin to SQL, and so on).

We categorize the examples in this section into four categories based on whether the query includes *join* and/or *group by* operations. All examples in this section are based on the weather database that consists of the following two tables:
Stations(stationID, zipcode,latitude,longitude) and *Reports (stationID, temperature, humidity, year,month)*.

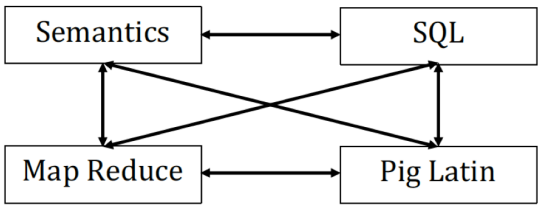

Figure 2: Different Representations of the Same Query.

3.1 Type 1: no *join*, no *group by*

A query in this group includes only *select* (σ) and *project*(π) operators. Based on Table 2, in map-reduce, this query is implemented by a map-reduce job where the *map* function performs both the selection and projection operations and the *reduce* function is the identity reducer that just passes the input tuples to the output. In Pig Latin, this query is implemented by a *filter* statement followed by a *foreach* statement. An example of type 1 query is as follows.

- **Semantics:** List station identifiers for all stations located in zipcode 55112.
- **SQL:**
  - `select` $stationID$
  - `from` $Stations$
  - `where` $zipcode = 55112$
- **Map reduce:**
  - `map`: for every tuple in *Stations*, if *zipcode* equals to 55112, output $(stationID, 1)$.
  - `reduce`: identity reducer.
- **Pig Latin:**
  - $temp =$ `filter` $Stations$ `by` $zipcode = 55112$;
  - $output =$ `foreach` $temp$ `generate` $stationID$;

3.2 Type 2: *join*, no *group by*

A query in this group includes *select* (σ), *project* (π), and *join* ($\bowtie$) operations. In map-reduce, the *map* function performs the selection and projection operations and set the join attribute as the output *key*. The output *value* includes the attributes required by the projection operation. The actual join operation is performed in the *reduce* function. In Pig Latin, a query in this group needs three steps, *filter*, *foreach*, and then *join*. Note the similarities between Pig's sequence of statements and the query execution pipeline as explained in Section 2. An example of type 2 query is as follows.

- **Semantics:** List all stationID and temperature values that were reported in 2000 by all stations that are located in zipcode 55126.
- **SQL:**
  - `select` $stationID,\ temperature$
  - `from` $Stations\ S,\ Reports\ R$
  - `where` $S.stationID = R.stationID$ `and` $Year = 2000$ `and` $zipcode\ =\ 55126$
- **Map reduce:**
  - `map`: for each tuple in *Stations*, if *zipcode* equals 55126, output $(stationID, (1, zipcode))$ and for each tuple in *Reports*, if *year* equals to 2000, output $(stationID, (2, temperature))$
  - `reduce`: For every input $stationID$ key, search the values list for a tuple with the format, $(stationID, (1, zipcode))$. If not found, do not produce any output, else if found, for each value with the format $(2, temperature)$, produce in the output a tuple with the format $(stationID, temperature)$.
- **Pig Latin:**
  - $temp1\ =$ `filter` $Reports$ `by` $Year\ =\ 2000$;
  - $temp2\ =$ `filter` $Stations$ `by` $zipcode\ =\ 55126$;
  - $temp3\ =$ `join` $temp1$ `by` $stationID,\ temp2$ `by` $stationID$;
  - $output\ =$ `foreach` $temp3$ `generate` $temp1.stationID,\ temp1.temperature$

3.3 Type 3: no *join*, *group by*

A query in this group includes *select* (σ), *project* (π), and *group by* (γ) operations. In map-reduce, the *map* function performs the selection and projection operations and set the grouping attribute as the output *key*. The output *value* includes other attributes that are needed by the aggregate operations. The map-reduce framework automatically performs the grouping and then the *reduce* function performs the aggregation. In Pig Latin, a query in this group is performed by three steps, *filter*, *group* and *foreach*. Again, the order of Pig Latin's statements is the same as the order of operations in the SQL's query execution plan. An example of type 3 query is as follows.

- **Semantics:** Find the total number of stations in each zipcode that is located on latitude 90 or above.
- **SQL:**
  - `select` $zipcode,\ count(*)$
  - `from` $Stations$
  - `where` $latitude\ \geq\ 90$
  - `group by` $zipcode$

- **Map reduce:**
  - `map`: for each tuple in *Stations*, if *latitude* is greater than or equal to 90, output $(zipcode, 1)$
  - `reduce`: for every input *key* (which is a zipcode), let x equals to the number of ones in the input values list and output $(zipcode, x)$
- **Pig Latin:**
  - $temp1$ = `filter` $Stations$ `by` $latitude \geq 90$;
  - $temp2$ = `group` $temp1$ `by` $zipcode$;
  - $output$ = `foreach` $temp2$ `generate` $group$, $COUNT(temp1)$;

3.4 Type 4: *join*, *group by*

A query in this group includes *select* (σ), *project* (π), *join* ($\bowtie$), and *group by* (γ) operations. In map-reduce, a query in this group has to be implemented using a sequence of two map-reduce jobs as follows: (1) the first job performs selection, projection and join same as type 2 query, (2) the output of the first job is used as input for the second job, and (3) the second job performs the group by and aggregation operations same as type 3 query.

- **Semantics:** Find the maximum temperature that was reported for each zip code that is located at latitude 90.
- **SQL:**
  - `select` $zipcode$, $max(temperature)$
  - `from` $Stations\ S$, $Reports\ R$
  - `where` $S.stationID = R.statationID$ `and` $latitude = 90$
  - `group by` *by* $zipcode$
- **Map reduce:**
  - `map 1`:for each tuple in *Stations*, if *latitude* equals 90, output $(stationID, (1, zipcode))$ and for every tuple in *Reports* output $(stationID, (2, temperature))$
  - `reduce 1`: for every input $stationID$ key, search the values list for a tuple with the format, $(stationID, (1, zipcode))$. If not found, do not produce any output, else if found, for each value with the format $(2, temperature)$, produce in the output a tuple with the format $(stationID, temperature)$.
  - `map 2`: identity mapper that just passes every input $(stationID, temperature)$ to the output.
  - `reduce 2` : for each input $stationID$, let x be the maximum temperature value among the input list of values, then output $(stationID, x)$
- **Pig Latin:**
  - $temp1$ = `filter` $Stations$ `by` $latitude = 90$;
  - $temp2$ = `join` $temp1$ `by` $stationID$, $Reports$*by* $stationID$;
  - $temp3$ = `group` $temp2$ `by` $zipcode$;
  - $output$ = `foreach` $temp3$ `generate` $group, MAX(temp2.Reports :: temperature)$;

4 Programming Environment

Hands-on programming exercises represent a key factor in understanding programming languages. Unfortunately, due to the nature of the map-reduce framework, a computer cluster would be needed to directly enable such exercises. Many institutions, however, do not have such clusters available for teaching. One solution to this problem is to use a virtual machine (VM) that simulates a Hadoop cluster and includes all the required software components. One option for such virtual machine is the Cloudera QuickStart Virtual Machine that can be be download from [1]. Once downloaded, the VM can be run using a virtual machine player (e.g., VirtualBox or VMWare). Once the virtual machine is downloaded, running SQL queries and Pig Latin scripts is straightforward. For example, Figure 3 shows how *Impala* and *Pig Latin* editors can be opened from a browser window from inside the Cloudera Virtual machine.

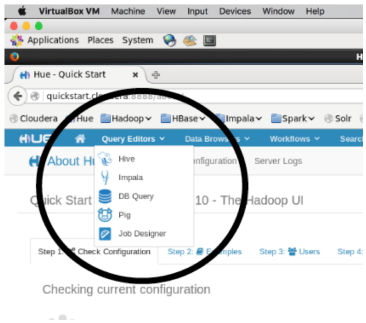

Figure 3: SQL and Pig Latin Editors.

Running map-reduce code is more challenging as students need to write and compile Java code. However, to facilitate the process of writing map-reduce programs, instructors can give students a map-reduce program template that imports all the required libraries, provides empty definitions of the *map* and *reduce* methods and includes a generic version of the driver that creates, configures, and runs a generic map-reduce job.

5 Conclusions

Several new programming languages have been introduced to implement applications that handles large data sets. In order to cope with he job market requirements, computer science students need to have a good understanding of the newly introduced technologies. A main challenge for computer science educators is to develop a well-thought-of categorization the new technologies and teach student the differences and similarities among technologies instead of teaching one or two specific technologies. In this paper, we proposed a categorization of big-data query languages into three categories: procedural, scripting, and declarative. We then provided teaching materials to teach the basic concepts of three big-data query languages concurrently by contrasting how the same relational-algebra query is represented by three syntactically-different languages. The materials introduced in this paper can be either taught as a part of an existing database course or can be expanded and taught as an introductory course on big data.

References

[1] Cloudera quickstat virtual machine. http://www.cloudera.com/downloads/quickstart.

[2] Sql: From traditional databases to big data - course resources - http://www.public.asu.edu/ ynsilva/ibigdata.

[3] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, 1970.

[4] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *6th Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, California, USA, December 6-8, 2004*, pages 137–150, 2004.

[5] J. D. U. Jure Leskovec, Anand Rajaraman. Mining of massive data sets. http://infolab.stanford.edu/ ullman/mmds/book.pdf.

[6] M. Kornacker, A. Behm, V. Bittorf, T. Bobrovytsky, C. Ching, A. Choi, J. Erickson, M. Grund, D. Hecht, M. Jacobs, I. Joshi, L. Kuff, D. Kumar, A. Leblang, N. Li, I. Pandis, H. Robinson, D. Rorke, S. Rus, J. Russell, D. Tsirogiannis, S. Wanderman-Milne, and M. Yoder. Impala: A modern, open-source SQL engine for hadoop. In *CIDR 2015, Seventh Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2015, Online Proceedings*, 2015.

[7] J. Lin and C. Dyer. Data-intensive text processing using map reduce https://lintool.github.io/mapreducealgorithms/mapreduce-book-final.pdf.

[8] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pages 1099–1110, 2008.

[9] B. Ramamurthy. A practical and sustainable model for learning and teaching data science. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education, Memphis, TN, USA, March 02 - 05, 2016*, pages 169–174, 2016.

[10] E. S. Richard Brown, Patrick Garrity. Teaching map-reduce parallel computing in cs1. *Midwest Instruction and Computing Symposium*, 2011.

[11] R. W. Sebesta. *Concepts of programming languages (4. ed.)*. Addison-Wesley-Longman, 1999.

[12] A. Silberschatz, H. F. Korth, and S. Sudarshan. *Database System Concepts, 5th Edition*. McGraw-Hill Book Company, 2005.

[13] Y. N. Silva, I. Almeida, and M. Queiroz. SQL: from traditional databases to big data. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education, Memphis, TN, USA, March 02 - 05, 2016*, pages 413–418, 2016.

[14] Y. N. Silva, S. W. Dietrich, J. M. Reed, and L. M. Tsosie. Integrating big data into the computing curricula. In *The 45th ACM Technical Symposium on Computer Science Education, SIGCSE '14, Atlanta, GA, USA - March 05 - 08, 2014*, pages 139–144, 2014.

[15] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Anthony, H. Liu, and R. Murthy. Hive - a petabyte scale data warehouse using hadoop. In *Proceedings of the 26th International Conference on Data Engineering, ICDE 2010, March 1-6, 2010, Long Beach, California, USA*, pages 996–1005, 2010.

[16] T. White. *Hadoop - The Definitive Guide: Storage and Analysis at Internet Scale (4. ed., revised & updated)*. O'Reilly, 2015.

[17] H. Yang, A. Dasdan, R. Hsiao, and D. S. P. Jr. Map-reduce-merge: simplified relational data processing on large clusters. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12-14, 2007*, pages 1029–1040, 2007.