US 20170091317A1

(19) **United States**

(12) **Patent Application Publication**

Cummings et al.

(10) **Pub. No.: US 2017/0091317 A1**

(43) **Pub. Date: Mar. 30, 2017**

(54) **LOCATION CORRELATION BETWEEN QUERY SCRIPT AND DATA FLOW**

(71) Applicant: **Microsoft Technology Licensing, LLC**, Redmond, WA (US)

(72) Inventors: **David Joseph Cummings**, Kirkland, WA (US); **Zhaoji Chen**, Issaquah, WA (US); **Yifung Lin**, Haidian District (CN); **Dian Zhang**, Haidian District (CN)

(21) Appl. No.: **15/195,657**

(22) Filed: **Jun. 28, 2016**

Related U.S. Application Data

(60) Provisional application No. 62/233,967, filed on Sep. 28, 2015.

Publication Classification

(51) **Int. Cl.**

G06F 17/30 (2006.01)

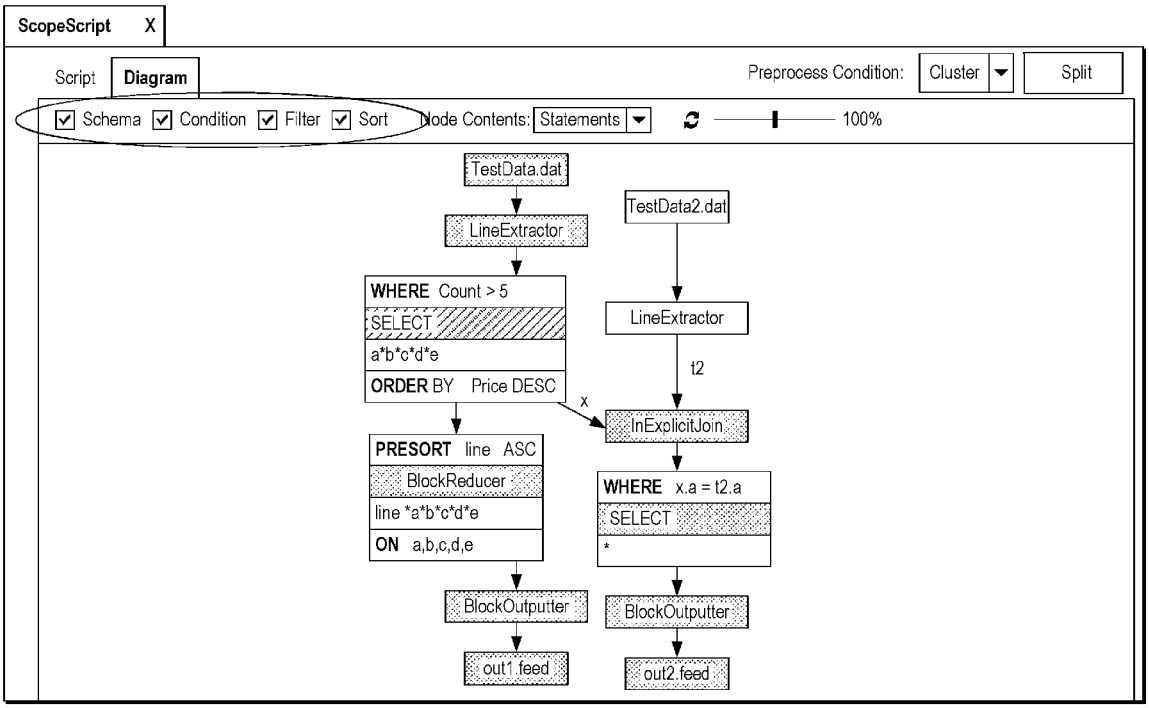
G06F 17/27 (2006.01)

(52) **U.S. Cl.**

CPC .. ***G06F 17/30684*** (2013.01); ***G06F 17/30625*** (2013.01); ***G06F 17/277*** (2013.01); ***G06F 17/271*** (2013.01)

(57) **ABSTRACT**

A computerized mechanism to automatically correlate positions of query script to portions of a data flow representation of the query script. When parsing the query script to generate the tokens, at least some of the tokens have an associated script location marker that identifies a location in the query script where the token originated from. The syntax tree of multiple nodes is then formulated, each node comprising one or more of the tokens parsed from the query script. Accordingly, the syntax tree retains the script location markers. A data flow representation of the query script is then formulated into a data flow representation. That data flow representation might, for instance, be based on the syntax tree, but augmented with data types of the various data flows. Nevertheless, the location marker is retained within the data flow representation.



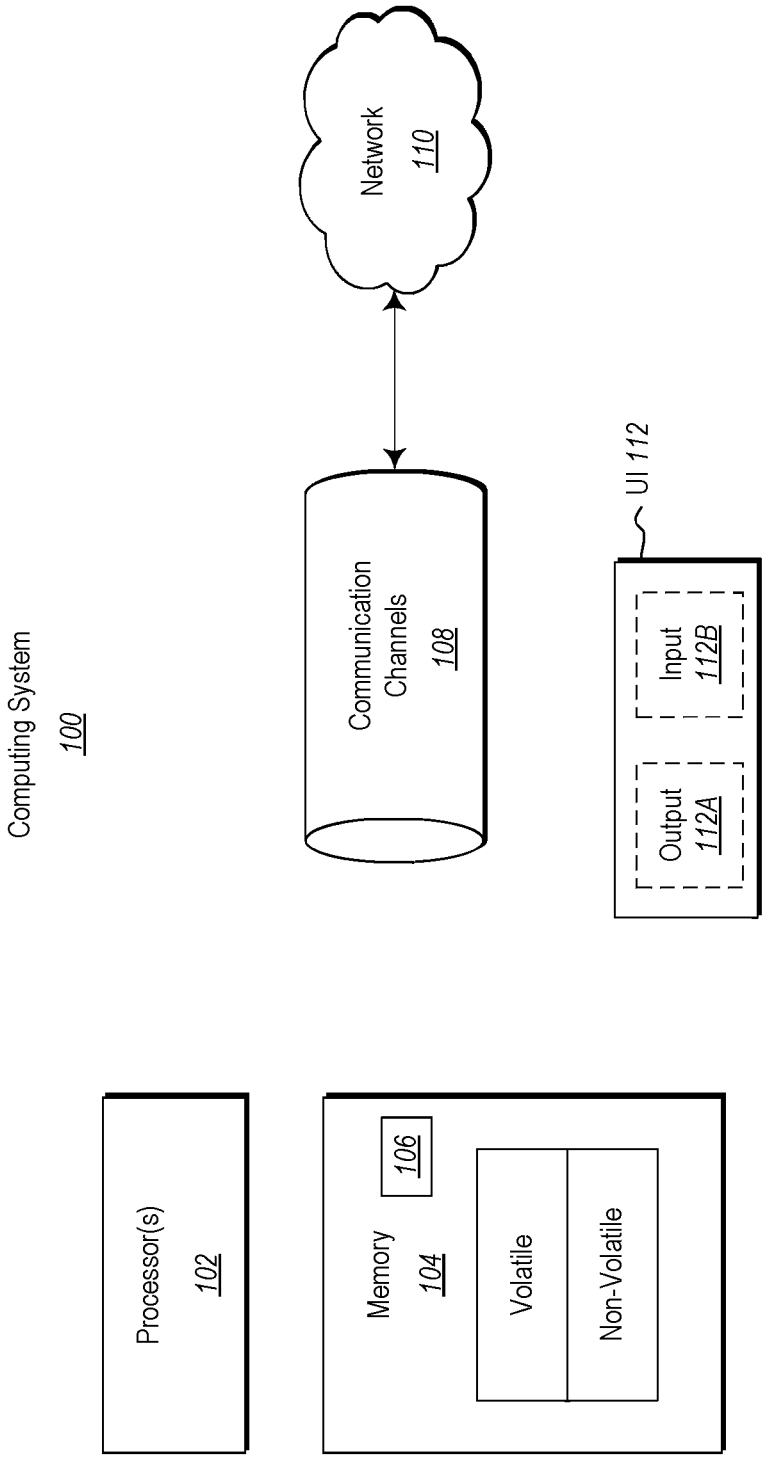

Computing System
100
Network
110
Communication Channels
108
UI 112
Output
112A
Input
112B
Processor(s)
102
Memory
104
106
Volatile
Non-Volatile

Figure 1

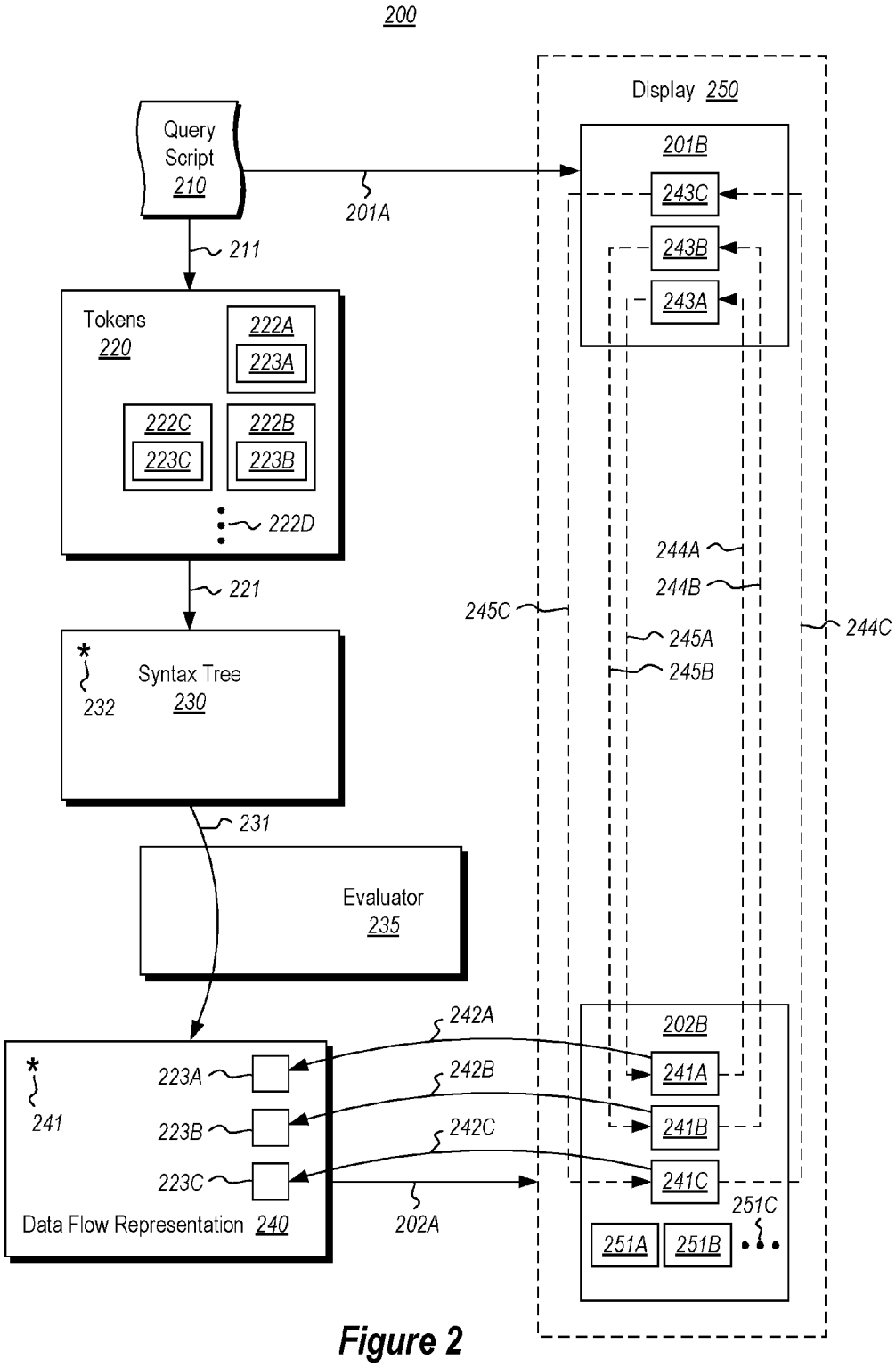

200
Display 250
Query Script 210
201A
201B
243C
243B
243A
211
Tokens 220
222A
223A
222C
223C
222B
223B
222D
221
244A
244B
245C
244C
245A
245B
Syntax Tree 230
232
231
Evaluator 235
242A
242B
242C
202B
241A
241B
241C
251C
251A
251B
223A
223B
223C
241
Data Flow Representation 240
202A

Figure 2

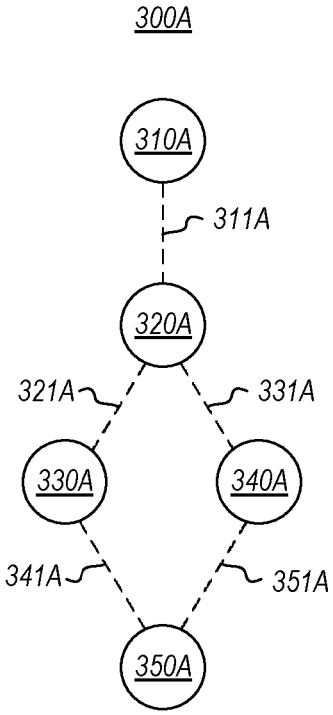


Figure 3A

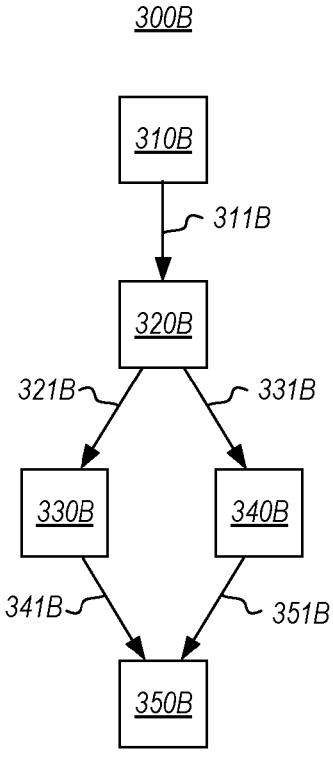


Figure 3B

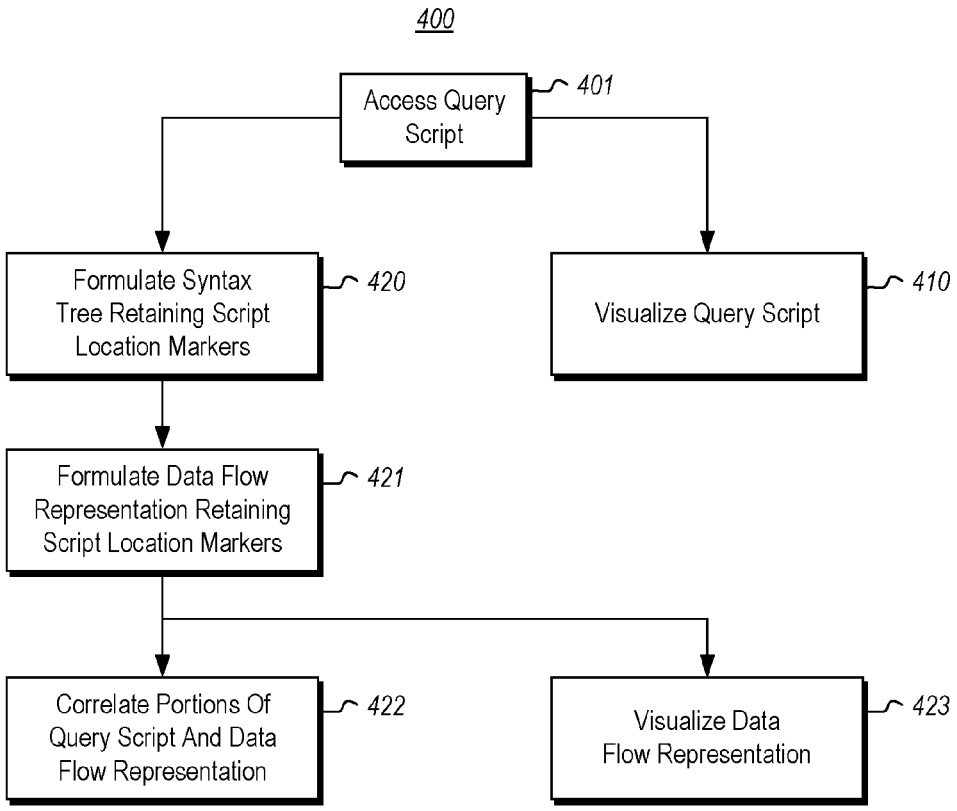


Figure 4

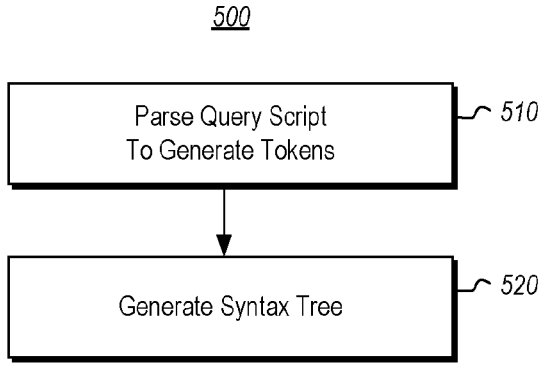


Figure 5

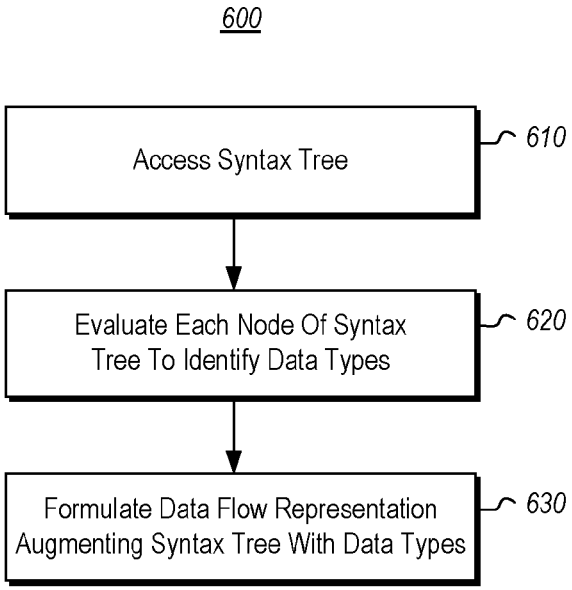


Figure 6

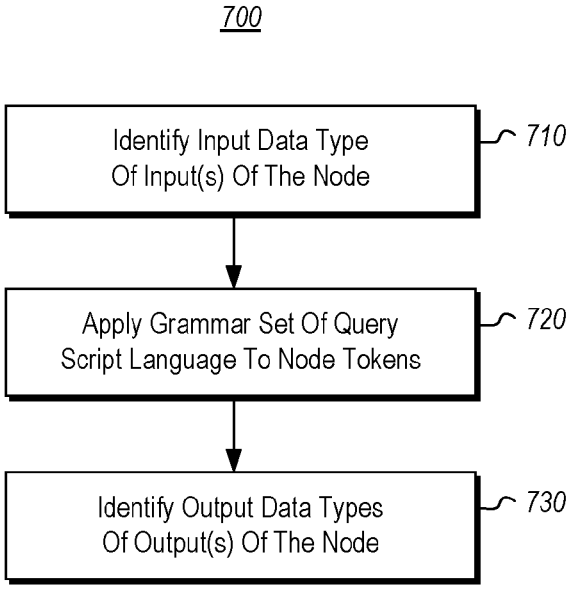


Figure 7

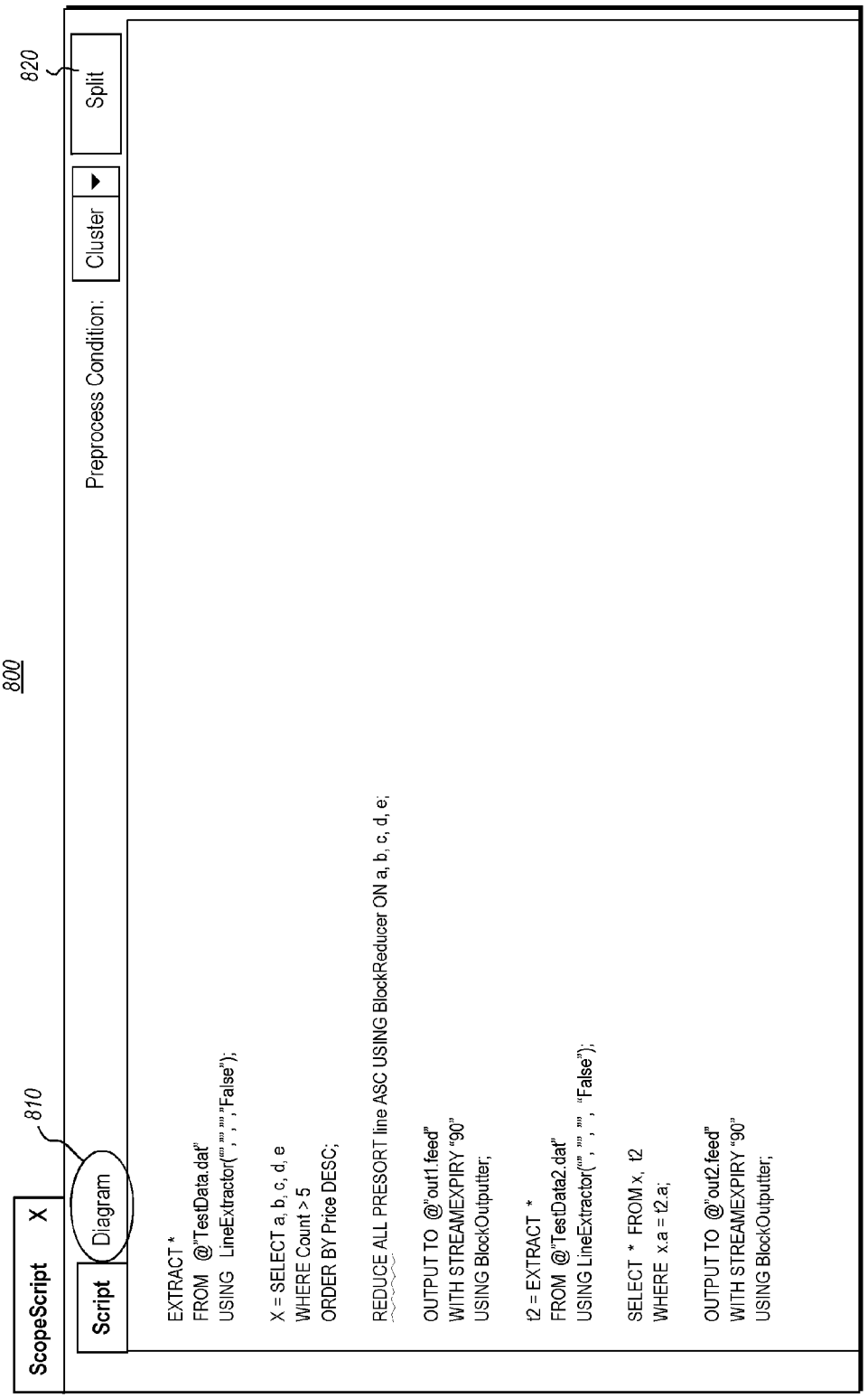

800
810
820
ScopeScript
X
Script
Diagram
Preprocess Condition:
Cluster
Split
EXTRACT *
FROM @"TestData.dat"
USING LineExtractor("", "", "", "False");
X = SELECT a, b, c, d, e
WHERE Count > 5
ORDER BY Price DESC;
REDUCE ALL PRESORT line ASC USING BlockReducer ON a, b, c, d, e;
OUTPUT TO @"out1.feed"
WITH STREAMEXPIRY "90"
USING BlockOutputter;
t2 = EXTRACT *
FROM @"TestData2.dat"
USING LineExtractor("", "", "", "False");
SELECT * FROM x, t2
WHERE x.a = t2.a;
OUTPUT TO @"out2.feed"
WITH STREAMEXPIRY "90"
USING BlockOutputter;

Figure 8

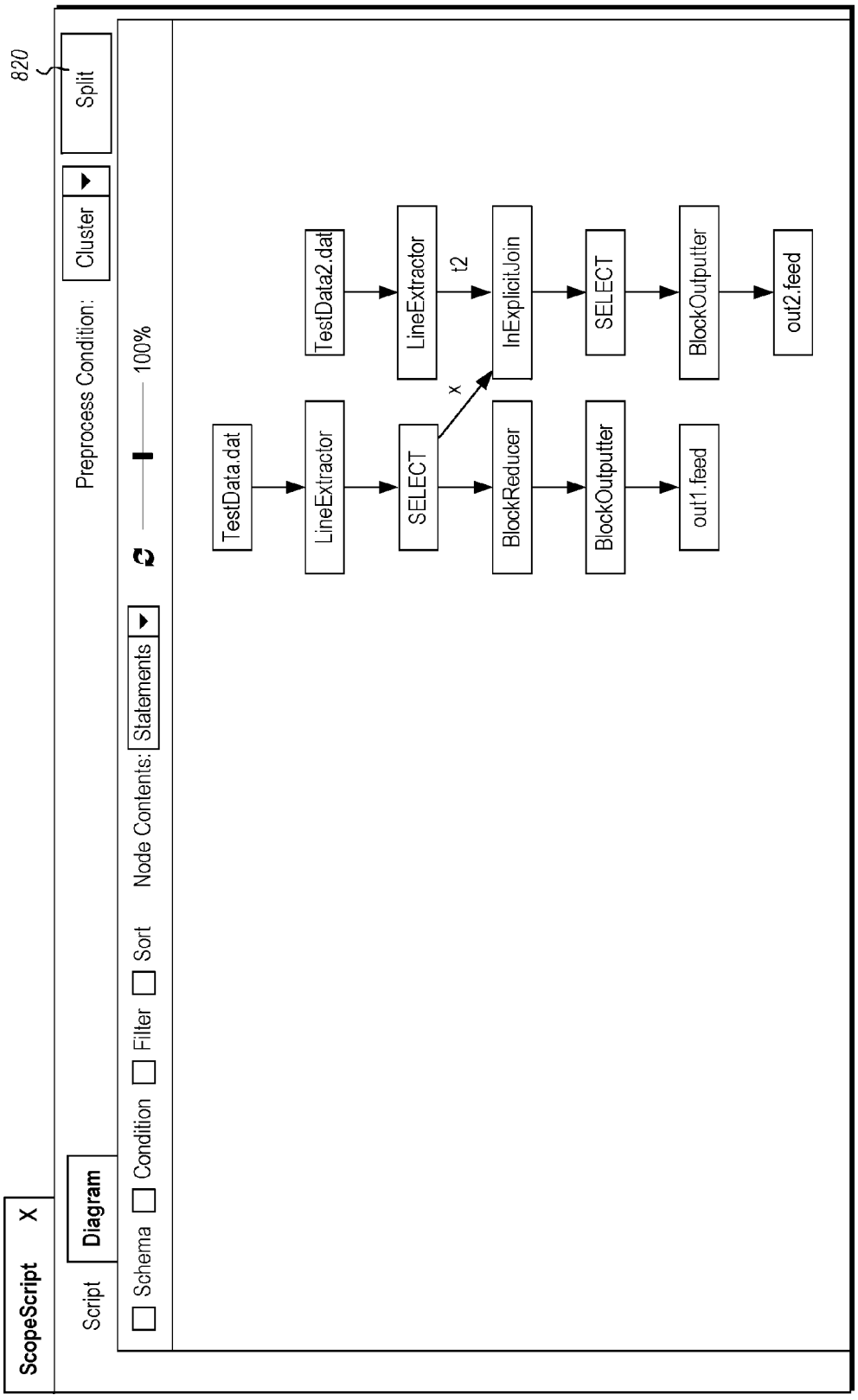

820
ScopeScript
X
Script
Diagram
Schema
Condition
Filter
Sort
Node Contents:
Statements
100%
Preprocess Condition:
Cluster
Split
TestData.dat
LineExtractor
SELECT
BlockReducer
BlockOutputter
out1.feed
TestData2.dat
LineExtractor
t2
x
InExplicitJoin
SELECT
BlockOutputter
out2.feed

Figure 9

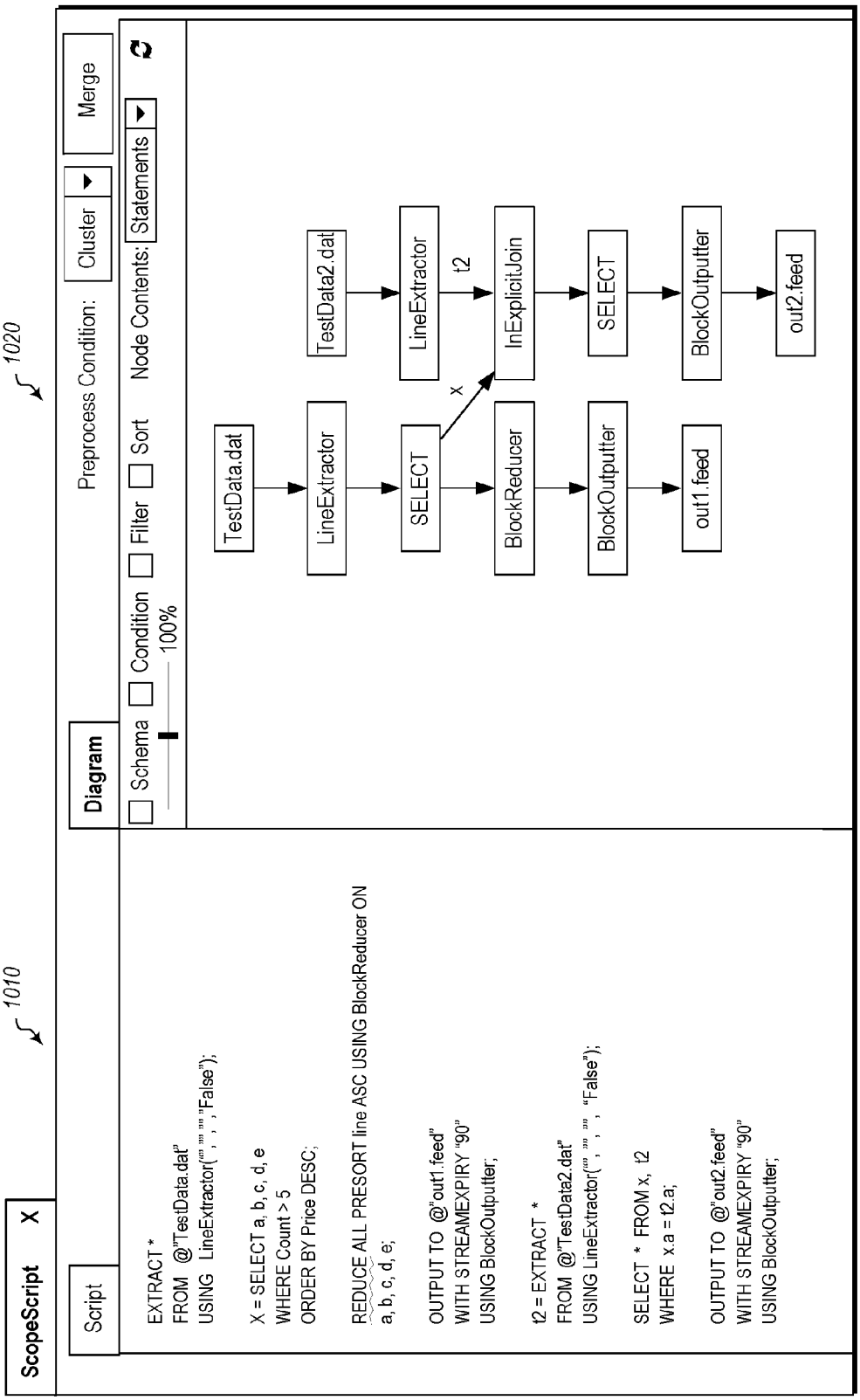

ScopeScript X
1010
1020
Script
Diagram
Schema
Condition
Filter
Sort
Preprocess Condition:
Cluster
Merge
Node Contents:
Statements
100%
EXTRACT *
FROM @"TestData.dat"
USING LineExtractor(",", "", "", "False");
X = SELECT a, b, c, d, e
WHERE Count > 5
ORDER BY Price DESC;
REDUCE ALL PRESORT line ASC USING BlockReducer ON
a, b, c, d, e;
OUTPUT TO @"out1.feed"
WITH STREAMEXPIRY "90"
USING BlockOutputter;
t2 = EXTRACT *
FROM @"TestData2.dat"
USING LineExtractor(",", "", "", "False");
SELECT * FROM x, t2
WHERE x.a = t2.a;
OUTPUT TO @"out2.feed"
WITH STREAMEXPIRY "90"
USING BlockOutputter;
TestData.dat
LineExtractor
SELECT
BlockReducer
BlockOutputter
out1.feed
TestData2.dat
LineExtractor
t2
x
InExplicitJoin
SELECT
BlockOutputter
out2.feed

Figure 10

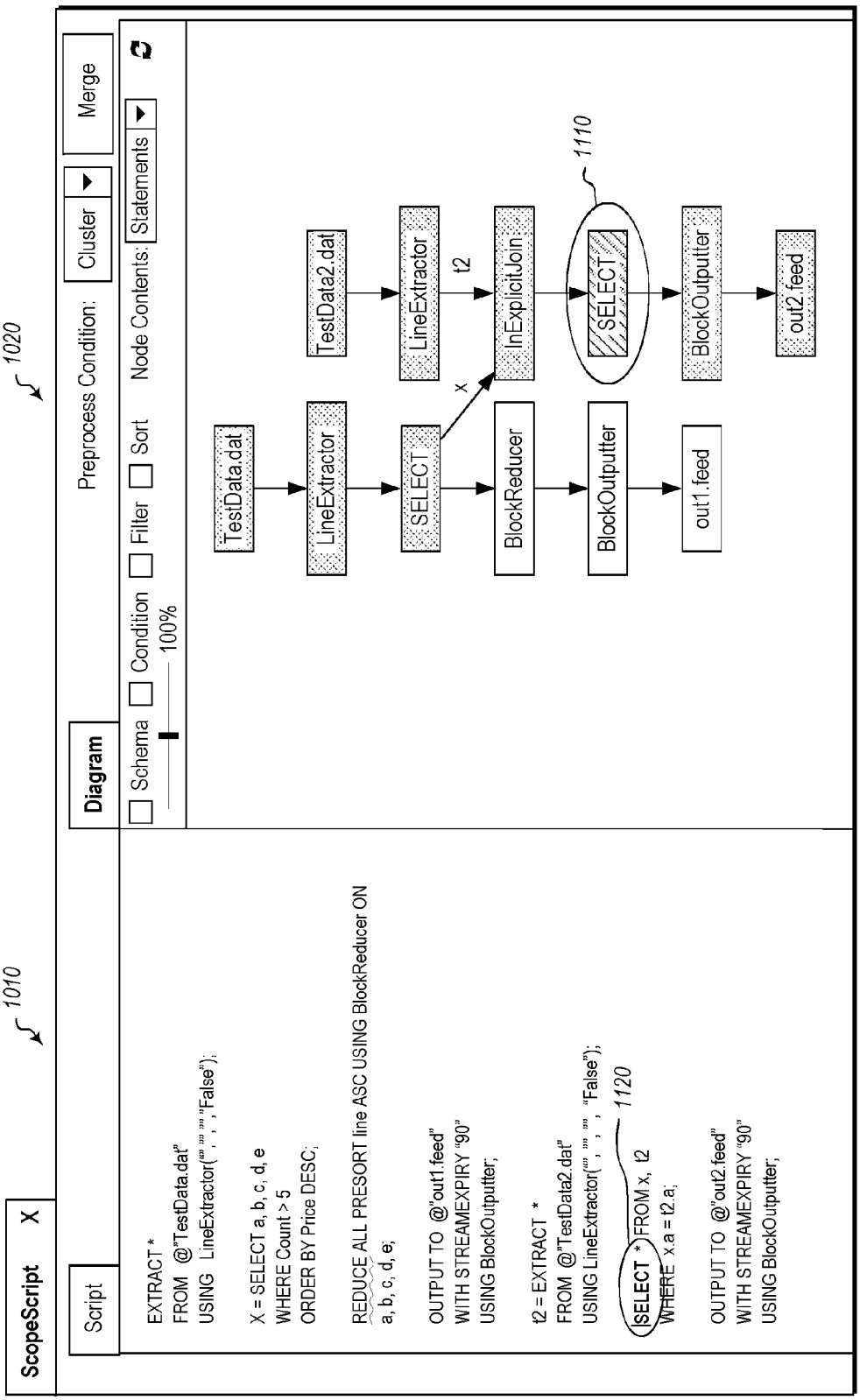

ScopeScript X
1010
1020
Script
Diagram
Preprocess Condition:
Cluster
Merge
Schema
Condition
Filter
Sort
Node Contents:
Statements
100%
EXTRACT *
FROM @"TestData.dat"
USING LineExtractor(" ", " ", " ", "False");
X = SELECT a, b, c, d, e
WHERE Count > 5
ORDER BY Price DESC;
REDUCE ALL PRESORT line ASC USING BlockReducer ON
a, b, c, d, e;
OUTPUT TO @"out1.feed"
WITH STREAMEXPIRY "90"
USING BlockOutputter;
t2 = EXTRACT *
FROM @"TestData2.dat"
USING LineExtractor(" ", " ", " ", "False");
SELECT * FROM x, t2
WHERE x.a = t2.a;
1120
OUTPUT TO @"out2.feed"
WITH STREAMEXPIRY "90"
USING BlockOutputter;
TestData.dat
LineExtractor
SELECT
x
BlockReducer
BlockOutputter
out1.feed
TestData2.dat
LineExtractor
t2
InExplicitJoin
SELECT
1110
BlockOutputter
out2.feed

Figure 11

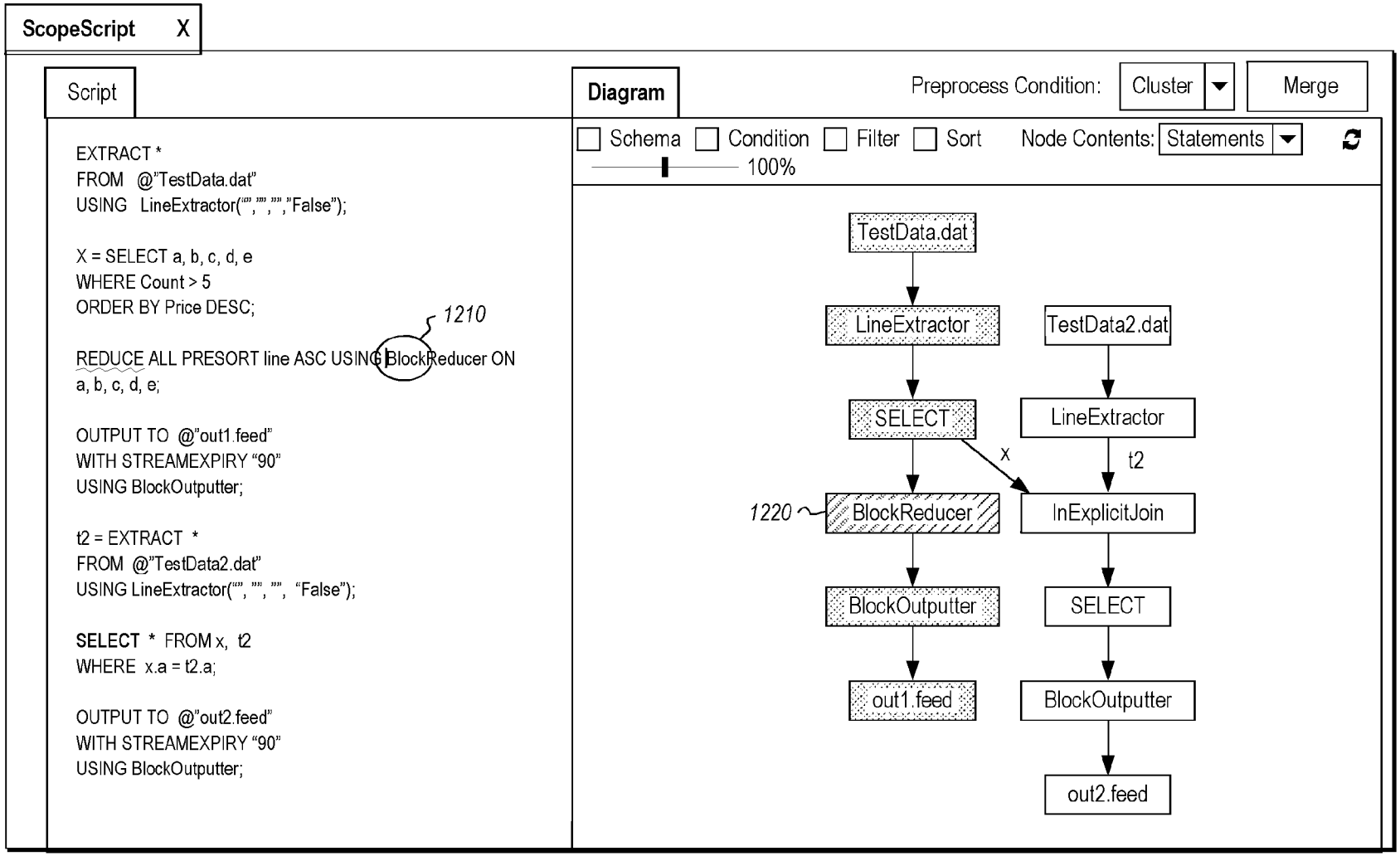

ScopeScript X
Script
EXTRACT *
FROM @"TestData.dat"
USING LineExtractor("","","","","False");
X = SELECT a, b, c, d, e
WHERE Count > 5
ORDER BY Price DESC;
1210
REDUCE ALL PRESORT line ASC USING BlockReducer ON
a, b, c, d, e;
OUTPUT TO @"out1.feed"
WITH STREAMEXPIRY "90"
USING BlockOutputter;
t2 = EXTRACT *
FROM @"TestData2.dat"
USING LineExtractor("", "", "", "False");
SELECT * FROM x, t2
WHERE x.a = t2.a;
OUTPUT TO @"out2.feed"
WITH STREAMEXPIRY "90"
USING BlockOutputter;
Diagram
Preprocess Condition:
Cluster
Merge
Schema
Condition
Filter
Sort
Node Contents:
Statements
100%
TestData.dat
LineExtractor
TestData2.dat
SELECT
LineExtractor
x
t2
1220
BlockReducer
InExplicitJoin
BlockOutputter
SELECT
out1.feed
BlockOutputter
out2.feed

Figure 12

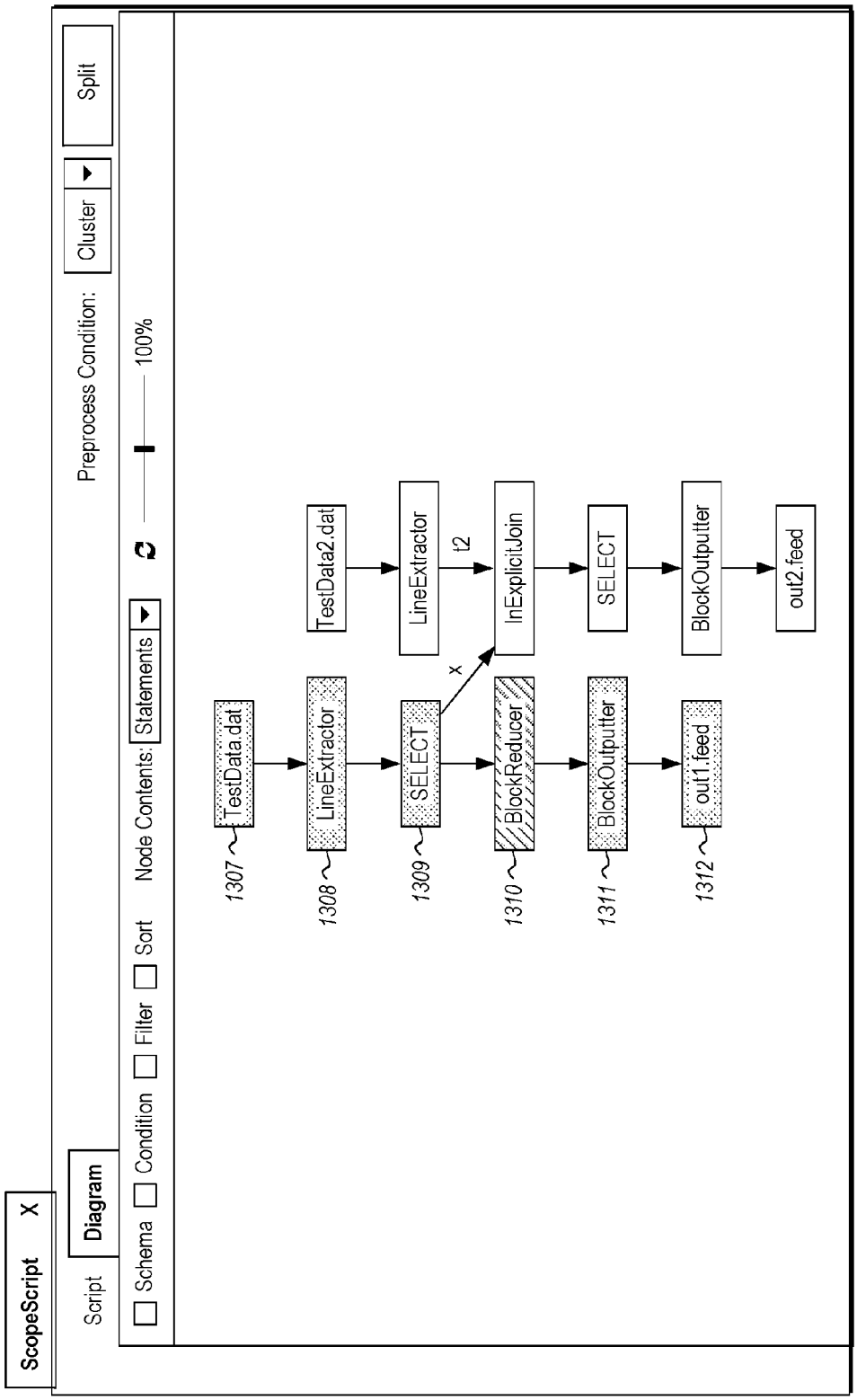

ScopeScript
Script
Diagram
Schema
Condition
Filter
Sort
Node Contents:
Statements
Preprocess Condition:
Cluster
Split
100%
TestData.dat
LineExtractor
SELECT
BlockReducer
BlockOutputter
out1.feed
TestData2.dat
LineExtractor
t2
InExplicitJoin
SELECT
BlockOutputter
out2.feed
x
1307
1308
1309
1310
1311
1312

Figure 13

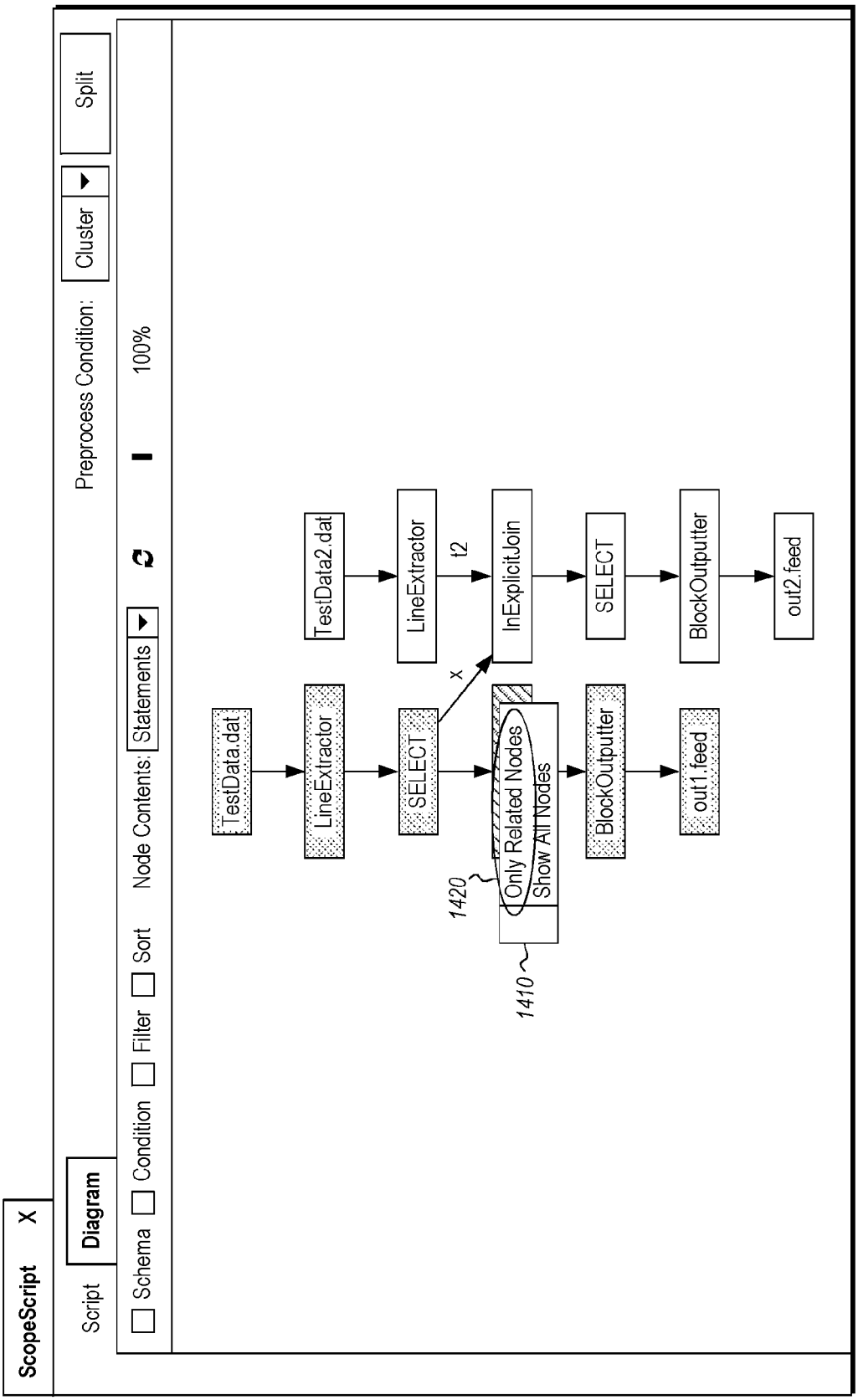

ScopeScript
X
Script
Diagram
Schema
Condition
Filter
Sort
Node Contents:
Statements
Preprocess Condition:
Cluster
Split
100%
TestData.dat
LineExtractor
SELECT
TestData2.dat
LineExtractor
t2
x
InExplicitJoin
SELECT
BlockOutputter
out2.feed
1420
Only Related Nodes
Show All Nodes
1410
BlockOutputter
out1.feed

Figure 14

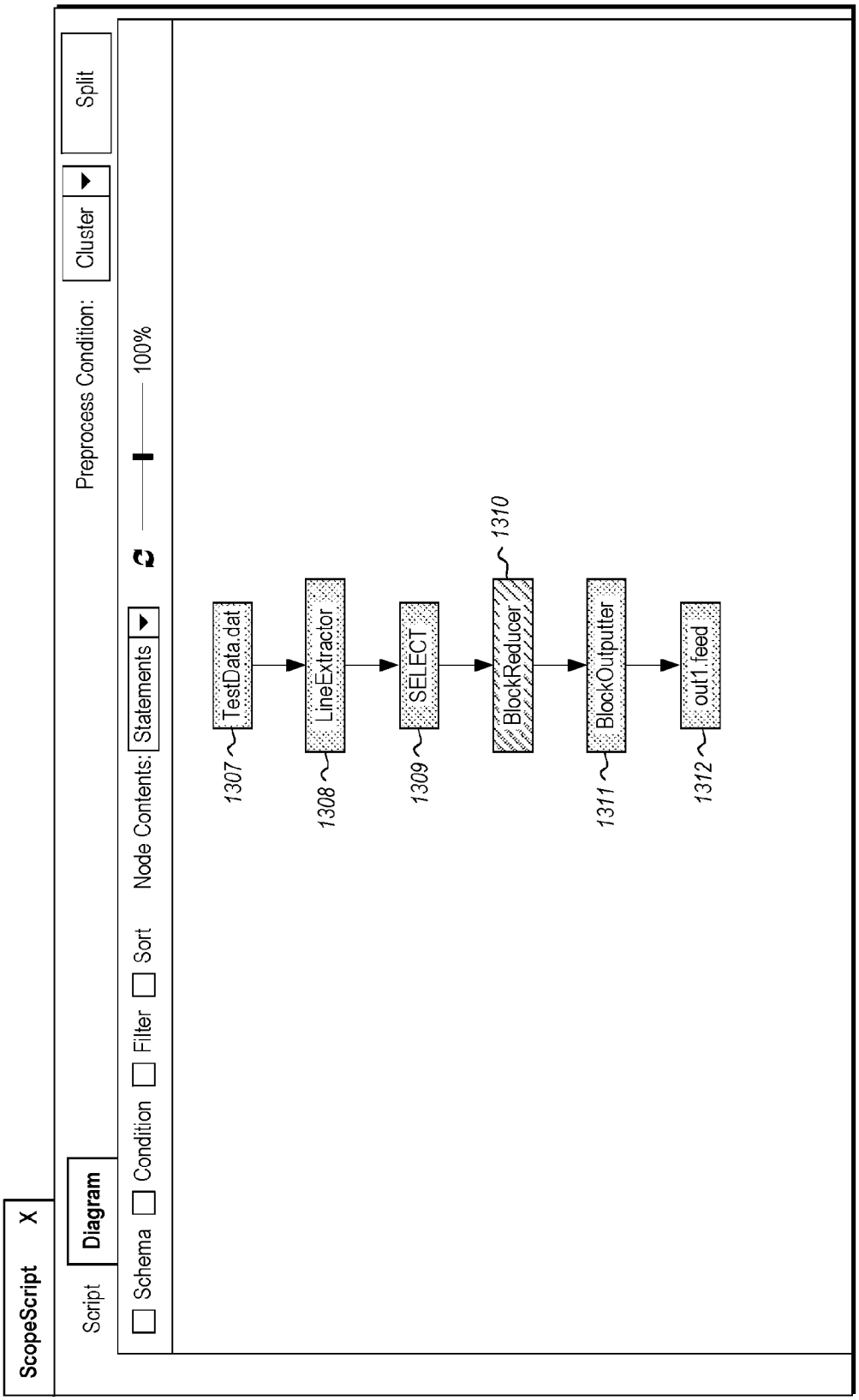

ScopeScript
Script
Diagram
Schema
Condition
Filter
Sort
Node Contents:
Statements
100%
Preprocess Condition:
Cluster
Split
TestData.dat
LineExtractor
SELECT
BlockReducer
BlockOutputter
out1.feed
1307
1308
1309
1310
1311
1312

Figure 15

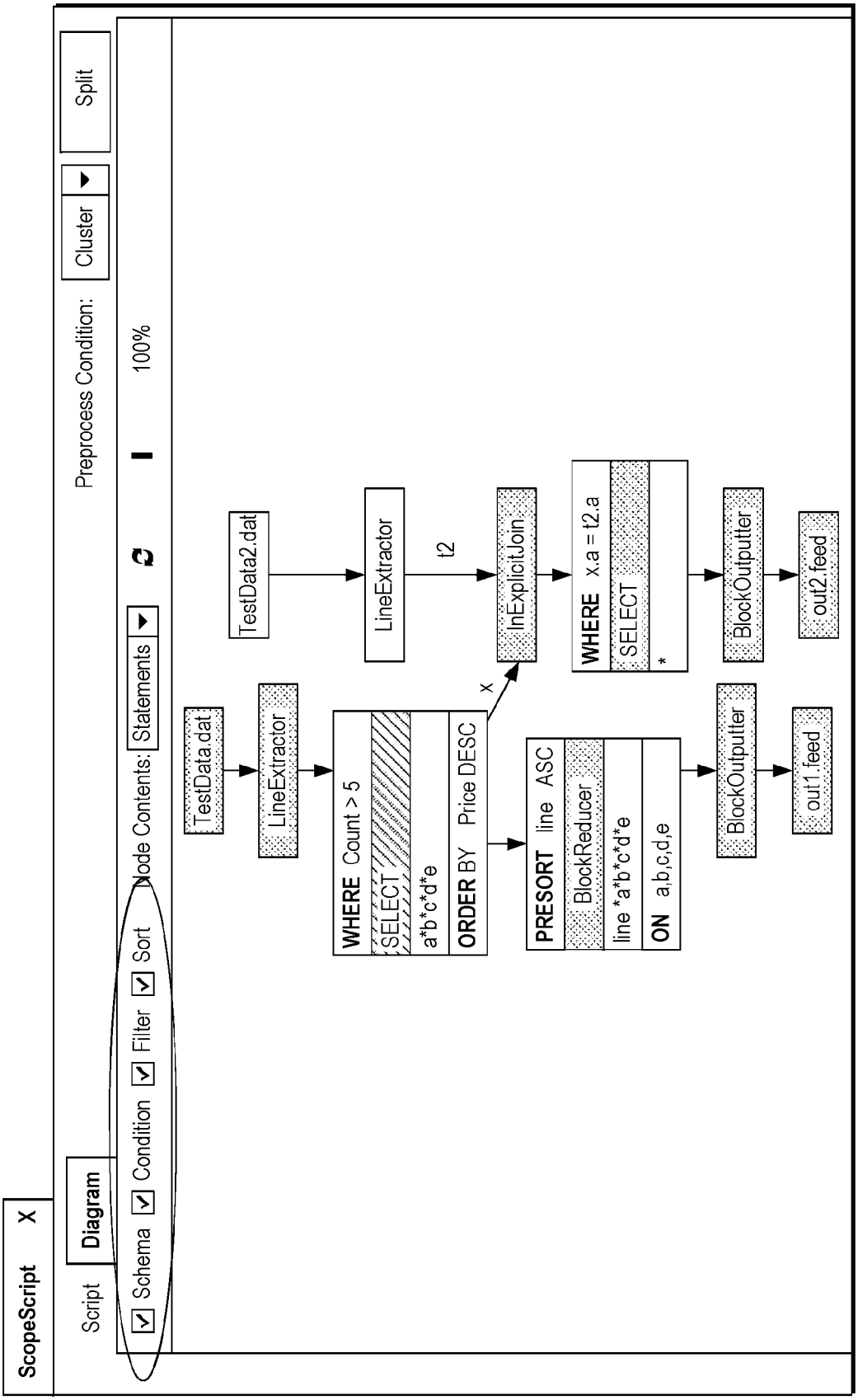

ScopeScript
Script
Diagram
Schema
Condition
Filter
Sort
Node Contents:
Statements
100%
Preprocess Condition:
Cluster
Split
TestData.dat
LineExtractor
WHERE Count > 5
SELECT
a*b*c*d*e
ORDER BY Price DESC
PRESORT line ASC
BlockReducer
line *a*b*c*d*e
ON a,b,c,d,e
BlockOutputter
out1.feed
TestData2.dat
LineExtractor
t2
x
InExplicitJoin
WHERE x.a = t2.a
SELECT
*
BlockOutputter
out2.feed

Figure 16

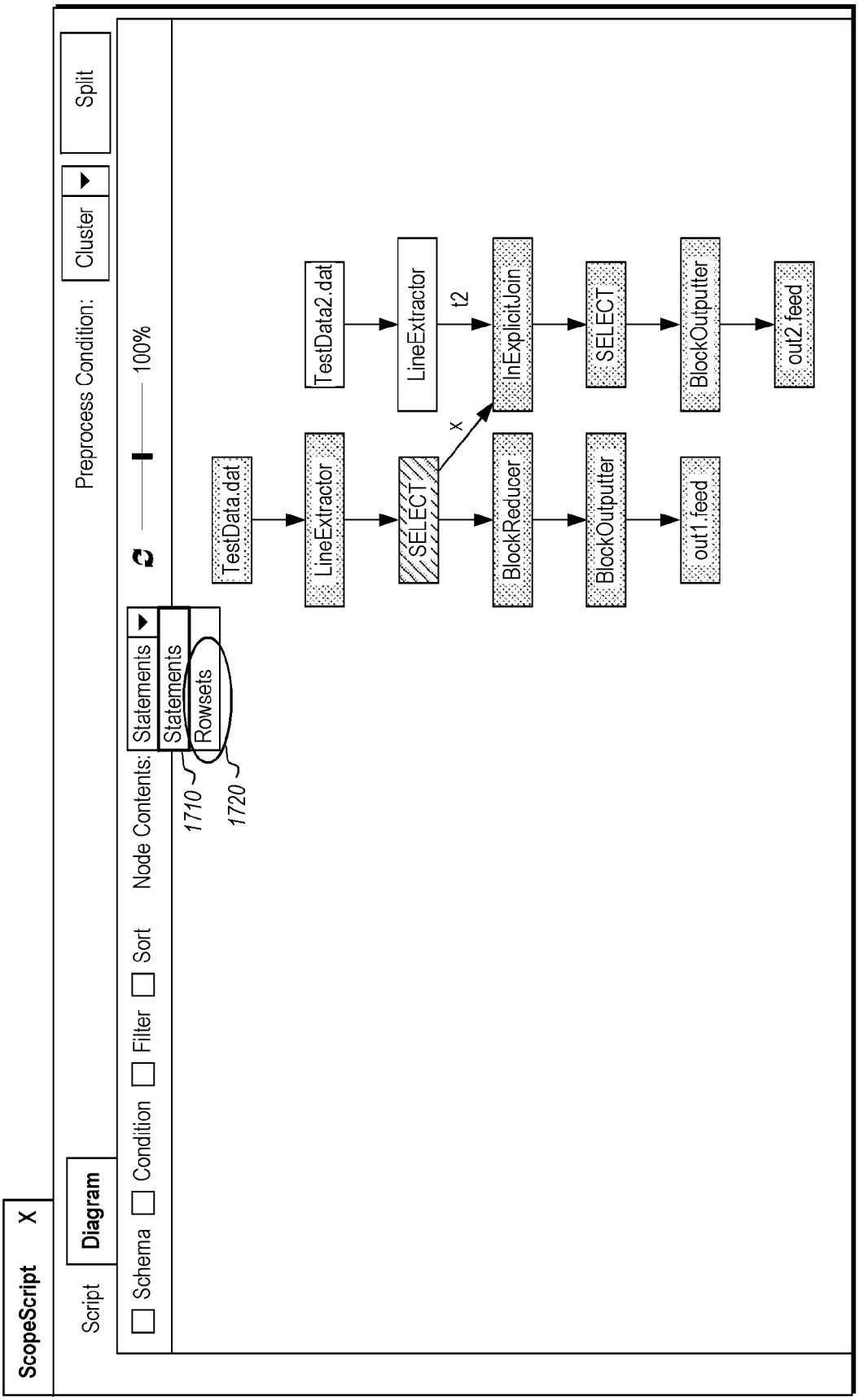

ScopeScript
Script
Diagram
Schema
Condition
Filter
Sort
Node Contents:
Statements
Statements
Rowsets
1710
1720
100%
Preprocess Condition:
Cluster
Split
TestData.dat
LineExtractor
SELECT
BlockReducer
BlockOutputter
out1.feed
x
TestData2.dat
LineExtractor
t2
InExplicitJoin
SELECT
BlockOutputter
out2.feed

Figure 17

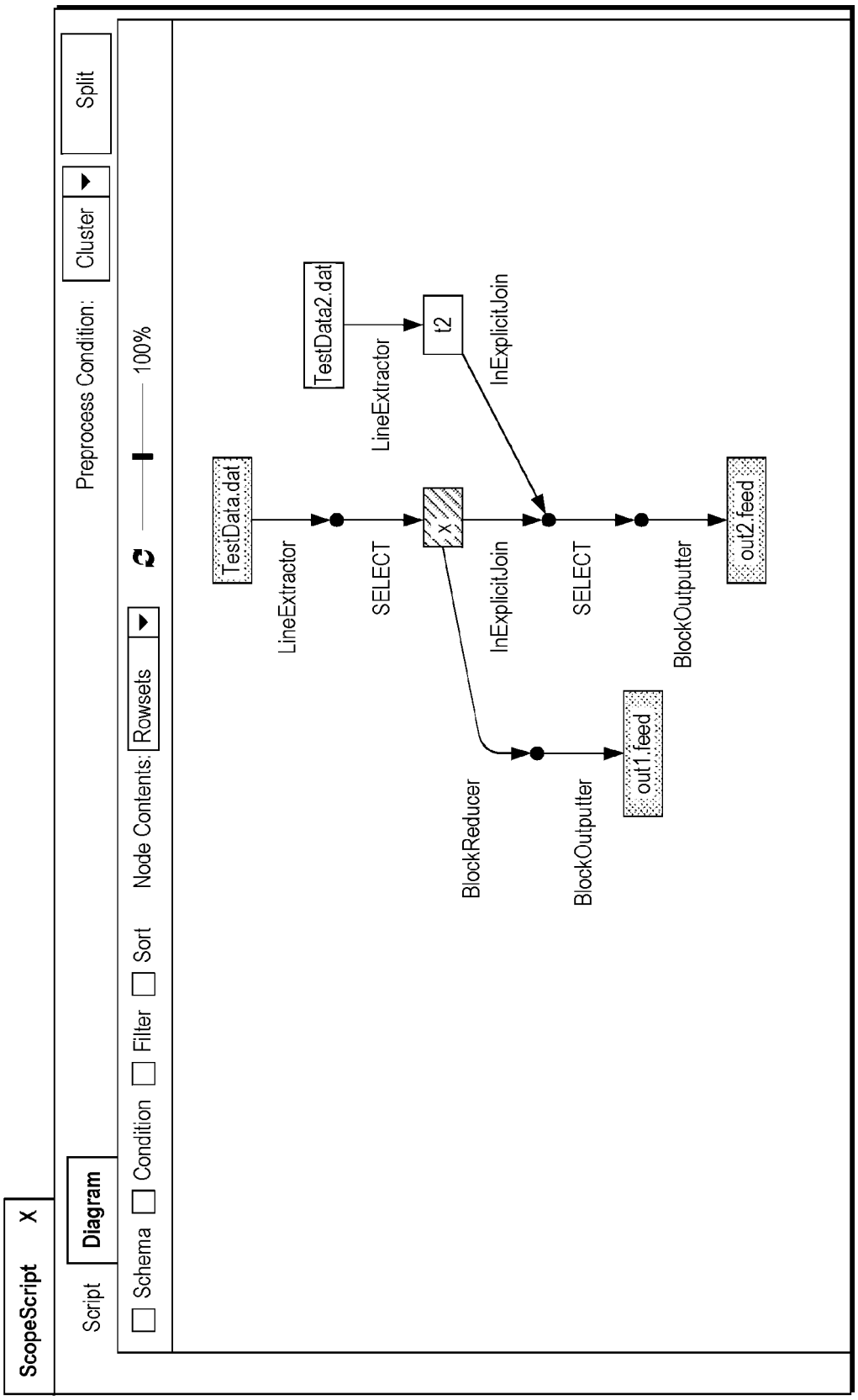

ScopeScript
Script
Diagram
Schema
Condition
Filter
Sort
Node Contents:
Rowsets
100%
Preprocess Condition:
Cluster
Split
TestData.dat
LineExtractor
SELECT
TestData2.dat
LineExtractor
t2
InExplicitJoin
BlockReducer
BlockOutputter
out1.feed
InExplicitJoin
SELECT
BlockOutputter
out2.feed

Figure 18

LOCATION CORRELATION BETWEEN QUERY SCRIPT AND DATA FLOW

CROSS-REFERENCE TO RELATED APPLICATION

[0001] This application claims the benefit of U.S. Provisional Patent Application Ser. No. 62/233,967 filed Sep. 28, 2015, which provisional patent application is incorporated herein by reference in its entirety.

BACKGROUND

[0002] Computing systems and associated networks have revolutionized the way human beings work, play, and communicate. Nearly every aspect of our lives is affected in some way by computing systems. More recently cloud computing has enabled users to offload much of the processing, storage, network I/O, memory, and other resource usage to various datacenters. This offloading of hardware capability is often referred to as Infrastructure As A Service (IAAS). Datacenters can also provide Platforms As A Service (PAAS), and even Software As a Service (SAAS). Since the users themselves typically do not have to be concerned about which datacenter or computing system are providing such hardware and software, the user is now able to be less concerned about the location of the hardware that is supporting the service, or how the services are being accessed. To the user, it is as though the user is simply reaching up into the nearest cloud or portion of the sky to obtain the desired computing service. The service seems ever present.

[0003] With data now often being moved into the cloud, the ability to store large quantities of data has improved greatly, enabling a technology field often referred to simply as "Big Data". For instance, big data queries may be processed against very large quantities of data, and those queries are efficiently processed in the cloud computing environment, allowing rapid return of results. Big data queries, like normal database queries, are typically declarative in form and are often referred to as "query script" or "script". There currently exist a variety of languages in which big data queries may be authored. When queries are processed, they are first parsed into tokens, and then the grammar set appropriate for the script language is then used to construct a syntax tree (also sometimes referred to as an "Abstract Syntax Tree" or AST).

[0004] The subject matter claimed herein is not limited to embodiments that solve any disadvantages or that operate only in environments such as those described above. Rather, this background is only provided to illustrate one exemplary technology area where some embodiments described herein may be practiced.

BRIEF SUMMARY

[0005] At least one embodiment described herein relates to a computerized mechanism to correlate positions of query script to portions of a data flow representation of the query script. Visualizations of such correlation would allow an author or reviewer of the query script to be able to quickly see what portions of the query script are going to cause what data flows. This gives the viewer and intuitive view on the operation of the query script.

[0006] When parsing the query script to generate the tokens, at least some of the tokens have an associated script location marker that identifies a location in the query script where the token originated from. For instance, the script location marker might be a line identifier of the query script. The syntax tree of multiple nodes is then formulated, each node comprising one or more of the tokens parsed from the query script. Accordingly, the syntax tree retains the script location markers. A data flow representation of the query script is then formulated into a data flow representation. That data flow representation might, for instance, be based on the syntax tree, but augmented with data types of the various data flows. Nevertheless, the script location marker is retained within the data flow representation.

[0007] Accordingly, when the data flow representation is visualized, each visualized node can be shown with its corresponding script portion emphasized. For instance, the data flow representation might be visualized on one half of a display, and the query script shown on the other half of the display. When a portion of the query script is selected, the corresponding portion of the visualized data flow portion may likewise be emphasized to show this correlation. Likewise, when a portion of the data flow representation is selected, the corresponding portion of the query script may be emphasized to show the correlation. This allows for more intuitive drafting and review of query script through computerized correlation of locations with the query script.

[0008] This Summary is not intended to identify key features or essential features of the claimed subject matter, nor is it intended to be used as an aid in determining the scope of the claimed subject matter.

BRIEF DESCRIPTION OF THE DRAWINGS

[0009] In order to describe the manner in which the above-recited and other advantages and features can be obtained, a more particular description of various embodiments will be rendered by reference to the appended drawings. Understanding that these drawings depict only sample embodiments and are not therefore to be considered to be limiting of the scope of the invention, the embodiments will be described and explained with additional specificity and detail through the use of the accompanying drawings in which:

[0010] FIG. **1** illustrates an example computing system in which the principles described herein may be employed;

[0011] FIG. **2** illustrates a flow representing a process for correlating query script to portions of a data flow representation of that query script in accordance with the principles described herein;

[0012] FIG. **3**A represents an example of a syntax tree that is used throughout the description;

[0013] FIG. **3**B represents an example of a data flow representation corresponding to the syntax tree of FIG. **3**A;

[0014] FIG. **4** illustrates a flowchart of a method for correlating positions of query script to portions of a data flow representation of the query script;

[0015] FIG. **5** illustrates a flowchart of an example method for formulating the syntax tree and represents an example of the act **420** of FIG. **4**.

[0016] FIG. **6** illustrates a flowchart of a method for generating a data flow representation from a syntax tree and represents an example of act **421** in FIG. **4**;

[0017] FIG. **7** illustrates a flowchart of a method for evaluating a node of the syntax tree and represents one example of how the act **620** of FIG. **6** may be performed;

[0018] FIGS. **8** through **18** show various user interface that are used to describe a specific user experience enabled by the principles described herein, and is provided for purposes of example only;

[0019] FIG. **8** illustrates a user interface in which only the query script is illustrated;

[0020] FIG. **9** illustrates the resulting user interface in this example user experience in which the user switches to a data flow representation view that shows the data flow representation of the query script shown in FIG. **8**;

[0021] FIG. **10** illustrates the split user interface that shows both the script view (showing the query script of FIG. **8**) and a data flow representation view (showing the data flow representation of FIG. **9**);

[0022] FIG. **11** is similar to FIG. **10**, except that the user has selected the node circled on the data flow representation view, resulting in a corresponding token being visually emphasized in the script view;

[0023] FIG. **12** is similar to FIG. **10**, except that the user has selected a token in the script view, resulting in a corresponding node being visually emphasized in the data flow representation view;

[0024] FIG. **13** illustrates a user interface in which the upstream and downstream nodes of a selected node is emphasized in the data flow representation view;

[0025] FIG. **14** shows a user interface in which a user may select to show only related nodes in the data flow representation view;

[0026] FIG. **15** shows a resulting user interface after the user selects to show only the related nodes in FIG. **14**;

[0027] FIG. **16** illustrates a user interface in which there are various options provided for hiding and showing parameters associated with the data flow representation view;

[0028] FIG. **17** illustrates a user interface in which the user may switch the roles of nodes and edges in the data flow representation view; and

[0029] FIG. **18** illustrates a user interface in which the user has switched the roles of nodes (now representing rowsets) and edges (now representing statements) in the data flow representation view.

DETAILED DESCRIPTION

[0030] At least one embodiment described herein relates to a computerized mechanism to correlate positions of query script to portions of a data flow representation of the query script. Visualizations of such correlation would allow an author or reviewer of the query script to be able to quickly see what portions of the query script are going to cause what data flows. This gives the viewer and intuitive view on the operation of the query script.

[0031] When parsing the query script to generate the tokens, at least some of the tokens have an associated script location marker that identifies a location in the query script where the token originated from. For instance, the script location marker might be a line identifier of the query script. The syntax tree of multiple nodes is then formulated, each node comprising one or more of the tokens parsed from the query script. Accordingly, the syntax tree retains the script location markers. A data flow representation of the query script is then formulated into a data flow representation. That data flow representation might, for instance, be based on the syntax tree, but augmented with data types of the various data flows. Nevertheless, the script location marker is retained within the data flow representation.

[0032] Accordingly, when the data flow representation is visualized, each visualized node can be shown with its corresponding script portion emphasized. For instance, the data flow representation might be visualized on one half of a display, and the query script shown on the other half of the display. When a portion of the query script is selected, the corresponding portion of the visualized data flow portion may likewise be emphasized to show this correlation. Likewise, when a portion of the data flow representation is selected, the corresponding portion of the query script may be emphasized to show the correlation. This allows for more intuitive drafting and review of query script through computerized correlation of locations with the query script.

[0033] Some introductory discussion of a computing system will be described with respect to FIG. **1**. Then, the general structure and operation of a mechanism to correlate positions of query script to portions of a data flow representation of the query script will be described with respect to FIGS. **2** through **7**. Finally, a specific example user experience will be described with respect to the user interfaces illustrated in FIGS. **8** through **18**.

[0034] Computing systems are now increasingly taking a wide variety of forms. Computing systems may, for example, be handheld devices, appliances, laptop computers, desktop computers, mainframes, distributed computing systems, datacenters, or even devices that have not conventionally been considered a computing system, such as wearables (e.g., glasses). In this description and in the claims, the term "computing system" is defined broadly as including any device or system (or combination thereof) that includes at least one physical and tangible processor, and a physical and tangible memory capable of having thereon computer-executable instructions that may be executed by a processor. The memory may take any form and may depend on the nature and form of the computing system. A computing system may be distributed over a network environment and may include multiple constituent computing systems.

[0035] As illustrated in FIG. **1**, in its most basic configuration, a computing system **100** typically includes at least one hardware processing unit **102** and memory **104**. The memory **104** may be physical system memory, which may be volatile, non-volatile, or some combination of the two. The term "memory" may also be used herein to refer to non-volatile mass storage such as physical storage media. If the computing system is distributed, the processing, memory and/or storage capability may be distributed as well.

[0036] The computing system **100** also has thereon multiple structures often referred to as an "executable component". For instance, the memory **104** of the computing system **100** is illustrated as including executable component **106**. The term "executable component" is the name for a structure that is well understood to one of ordinary skill in the art in the field of computing as being a structure that can be software, hardware, or a combination thereof. For instance, when implemented in software, one of ordinary skill in the art would understand that the structure of an executable component may include software objects, routines, methods that may be executed on the computing system, whether such an executable component exists in the heap of a computing system, or whether the executable component exists on computer-readable storage media.

[0037] In such a case, one of ordinary skill in the art will recognize that the structure of the executable component exists on a computer-readable medium such that, when

interpreted by one or more processors of a computing system (e.g., by a processor thread), the computing system is caused to perform a function. Such structure may be computer-readable directly by the processors (as is the case if the executable component were binary). Alternatively, the structure may be structured to be interpretable and/or compiled (whether in a single stage or in multiple stages) so as to generate such binary that is directly interpretable by the processors. Such an understanding of example structures of an executable component is well within the understanding of one of ordinary skill in the art of computing when using the term "executable component".

[0038] The term "executable component" is also well understood by one of ordinary skill as including structures that are implemented exclusively or near-exclusively in hardware, such as within a field programmable gate array (FPGA), an application specific integrated circuit (ASIC), or any other specialized circuit. Accordingly, the term "executable component" is a term for a structure that is well understood by those of ordinary skill in the art of computing, whether implemented in software, hardware, or a combination. In this description, the terms "component", "service", "engine", "module", "evaluator", "monitor", "scheduler", "manager", "module", "compiler", "virtual machine", "container", "environment" or the like may also be used. As used in this description and in the case, these terms (whether expressed with or without a modifying clause) are also intended to be synonymous with the term "executable component", and thus also have a structure that is well understood by those of ordinary skill in the art of computing.

[0039] In the description that follows, embodiments are described with reference to acts that are performed by one or more computing systems. If such acts are implemented in software, one or more processors (of the associated computing system that performs the act) direct the operation of the computing system in response to having executed computer-executable instructions that constitute an executable component. For example, such computer-executable instructions may be embodied on one or more computer-readable media that form a computer program product. An example of such an operation involves the manipulation of data.

[0040] The computer-executable instructions (and the manipulated data) may be stored in the memory **104** of the computing system **100**. Computing system **100** may also contain communication channels **108** that allow the computing system **100** to communicate with other computing systems over, for example, network **110**.

[0041] While not all computing systems require a user interface, in some embodiments, the computing system **100** includes a user interface **112** for use in interfacing with a user. The user interface **112** may include output mechanisms **112**A as well as input mechanisms **112**B. The principles described herein are not limited to the precise output mechanisms **112**A or input mechanisms **112**B as such will depend on the nature of the device. However, output mechanisms **112**A might include, for instance, speakers, displays, projectors, tactile output, valves, actuators, holograms, virtual reality environments, and so forth. Examples of input mechanisms **112**B might include, for instance, microphones, touchscreens, holograms, cameras, keyboards, accelerometers, levers, pedals, buttons, knobs, mouse of other pointer input, sensors of any type, a virtual reality environment, and so forth.

[0042] Embodiments described herein may comprise or utilize a special purpose or general-purpose computing system including computer hardware, such as, for example, one or more processors and system memory, as discussed in greater detail below. Embodiments described herein also include physical and other computer-readable media for carrying or storing computer-executable instructions and/or data structures. Such computer-readable media can be any available media that can be accessed by a general purpose or special purpose computing system. Computer-readable media that store computer-executable instructions are physical storage media. Computer-readable media that carry computer-executable instructions are transmission media. Thus, by way of example, and not limitation, embodiments of the invention can comprise at least two distinctly different kinds of computer-readable media: storage media and transmission media.

[0043] Computer-readable storage media includes RAM, ROM, EEPROM, CD-ROM or other optical disk storage, magnetic disk storage or other magnetic storage devices, or any other physical and tangible storage medium which can be used to store desired program code means in the form of computer-executable instructions or data structures and which can be accessed by a general purpose or special purpose computing system.

[0044] A "network" is defined as one or more data links that enable the transport of electronic data between computing systems and/or modules and/or other electronic devices. When information is transferred or provided over a network or another communications connection (either hardwired, wireless, or a combination of hardwired or wireless) to a computing system, the computing system properly views the connection as a transmission medium. Transmissions media can include a network and/or data links which can be used to carry desired program code means in the form of computer-executable instructions or data structures and which can be accessed by a general purpose or special purpose computing system. Combinations of the above should also be included within the scope of computer-readable media.

[0045] Further, upon reaching various computing system components, program code means in the form of computer-executable instructions or data structures can be transferred automatically from transmission media to storage media (or vice versa). For example, computer-executable instructions or data structures received over a network or data link can be buffered in RAM within a network interface module (e.g., a "NIC"), and then eventually transferred to computing system RAM and/or to less volatile storage media at a computing system. Thus, it should be understood that storage media can be included in computing system components that also (or even primarily) utilize transmission media.

[0046] Computer-executable instructions comprise, for example, instructions and data which, when executed at a processor, cause a general purpose computing system, special purpose computing system, or special purpose processing device to perform a certain function or group of functions. Alternatively, or in addition, the computer-executable instructions may configure the computing system to perform a certain function or group of functions. The computer executable instructions may be, for example, binaries or even instructions that undergo some translation (such as

compilation) before direct execution by the processors, such as intermediate format instructions such as assembly language, or even source code.

[0047] Although the subject matter has been described in language specific to structural features and/or methodological acts, it is to be understood that the subject matter defined in the appended claims is not necessarily limited to the described features or acts described above. Rather, the described features and acts are disclosed as example forms of implementing the claims.

[0048] Those skilled in the art will appreciate that the invention may be practiced in network computing environments with many types of computing system configurations, including, personal computers, desktop computers, laptop computers, message processors, hand-held devices, multi-processor systems, microprocessor-based or programmable consumer electronics, network PCs, minicomputers, mainframe computers, mobile telephones, PDAs, pagers, routers, switches, datacenters, wearables (such as glasses) and the like. The invention may also be practiced in distributed system environments where local and remote computing systems, which are linked (either by hardwired data links, wireless data links, or by a combination of hardwired and wireless data links) through a network, both perform tasks. In a distributed system environment, program modules may be located in both local and remote memory storage devices.

[0049] Those skilled in the art will also appreciate that the invention may be practiced in a cloud computing environment. Cloud computing environments may be distributed, although this is not required. When distributed, cloud computing environments may be distributed internationally within an organization and/or have components possessed across multiple organizations. In this description and the following claims, "cloud computing" is defined as a model for enabling on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services). The definition of "cloud computing" is not limited to any of the other numerous advantages that can be obtained from such a model when properly deployed.

[0050] For instance, cloud computing is currently employed in the marketplace so as to offer ubiquitous and convenient on-demand access to the shared pool of configurable computing resources. Furthermore, the shared pool of configurable computing resources can be rapidly provisioned via virtualization and released with low management effort or service provider interaction, and then scaled accordingly.

[0051] A cloud computing model can be composed of various characteristics such as on-demand self-service, broad network access, resource pooling, rapid elasticity, measured service, and so forth. A cloud computing model may also come in the form of various service models such as, for example, Software as a Service ("SaaS"), Platform as a Service ("PaaS"), and Infrastructure as a Service ("IaaS"). The cloud computing model may also be deployed using different deployment models such as private cloud, community cloud, public cloud, hybrid cloud, and so forth. In this description and in the claims, a "cloud computing environment" is an environment in which cloud computing is employed.

[0052] FIG. **2** illustrates a flow **200** representing a process for correlating query script to portions of a data flow representation of that query script in accordance with the principles described herein. The flow **200** begins with the accessing of a query script **210**. The query script **210** is drafted in accordance with a query language. In some embodiments, the query language is a big data query language. Examples of big data query languages include Hive query language, Spark SQL, BigQuery, although there are numerous other examples of big data query languages. The principles described herein are not limited to any particular big data query language, and are not limited to big data query languages at all. The query script may be visualized (as represented by arrow **201**A) into a visual representation **201**B that is output on a display **250**. For instance, if the process occurs on the computing system **100** of FIG. **1**, the query script may be visualized on a display represented as one of the output mechanisms **102**A.

[0053] The script query is first parsed (as represented by arrow **211**) into multiple tokens **220**. This may be performed by, for instance, the parser of a compiler. In this particular example, the tokens **220** are shown as including three tokens **222**A, **222**B and **222**C. However, the ellipses **222**D symbolically represent that the principles described herein is not limited to the number of tokens that query script is parsed into. A typical segment of query script will often have many more than three tokens.

[0054] One, some, or all of the tokens may have a corresponding script location marker that identifies what portion of the query script the token is located in or originated from. In this example, all of the tokens **222**A, **222**B and **222**C have a corresponding script location marker **223**A, **223**B and **223**C. For instance, the script location marker might be a line identifier for one or more lines, perhaps in conjunction with a horizontal offset position or range for each line. Accordingly, there is enough information within the script location marker to highlight or otherwise visually emphasize the token itself within with query script.

[0055] The collection of tokens **220** is then formulated (as represented by arrow **221**) into a syntax tree **230** comprising multiple nodes, each including one or more tokens. The formulation of tokens into syntax trees are known in the art and thus will not be described in detail herein. However, unlike conventional formulation of syntax trees, some or all of those script location markers remain associated with the tokens when the tokens are included within nodes of the syntax tree. This continued inclusion of the script location markers is represented by the syntax tree **230** including asterisk **232**.

[0056] The syntax tree **230** is then evaluated by an evaluator **235** to thereby generate (as represented by arrow **231**) a data flow representation **240** of the syntax tree **230**. The data flow representation **240** also continues to include the script location markers for the tokens as represented by the data flow representation **240** having asterisk **241**. For instance, the data flow representation **240** is illustrated in FIG. **2** as having the script location markers **223**A, **223**B and **223**C that are associated with the respective tokens **222**A, **222**B and **222**C. Accordingly, the script location markers **223**A, **223**B and **223**C are in the data flow representation **240** and remain associated with the original tokens **222**A, **222**B, **222**C. This allows positions and/or portions within the data flow representation to be correlated with positions in the query script using the appropriate script location marker.

[0057] As represented by arrow **202**A, a visualization **202**B of the data flow representation may be presented on

the display **250**. For instance, if the process occurs on the computing system **100** of FIG. **1**, the data flow visualization **202**B may be visualized on a display represented as one of the output mechanisms **112**A.

[0058] The visualization **202**B of the data flow representation is illustrated in FIG. **2** as including controls **241**A, **241**B and **241**C that are visually associated with visualizations **243**A, **243**B and **243**C of the respective tokens **222**A, **222**B and **222**C within the visualization **201**B. Each of the controls **241**A, **241**B and **241**C also has a reference (as represented by corresponding arrows **242**A, **242**B and **242**C) to the respective script location markers **223**A, **223**B and **223**C of the respective tokens **222**A, **222**B and **222**C.

[0059] Using this script location marker, when a control is selected in the visualization of the data flow, the appropriate portion (e.g., a token) of the visualization **201**B of the query script is highlighted. For instance, upon selecting control **241**A, a visualization **243**A of the token **222**A within the visualization **201**B of the query script **210** may be visually emphasized as represented by arrow **244**A. Upon selecting control **241**B, a visualization **243**B of the token **222**B within the visualization **201**B of the query script **210** may be visually emphasized as represented by arrow **244**B. Upon selecting control **241**C, a visualization **243**C of the token **222**C within the visualization **201**B of the query script **210** may be visually emphasized as represented by arrow **244**C.

[0060] In addition to representing tokens, the boxes **243**A, **243**B and **243**C within the visualization **201**B of the query script **210** may also themselves represent controls visually associated with the tokens. Furthermore, in addition to representing controls, the boxes **241**A, **241**B and **241**C within the visualization **202**B of the data flow representation **240** may also themselves represent visualized portions of the data flow representation that corresponding to the respective tokens **222**A, **222**B and **222**C. For instance, upon selecting control **243**A, a visualization **241**A of the node **222**A within the visualization **202**B of the data flow representation **240** may be visually emphasized as represented by arrow **245**A. Upon selecting control **243**B, a visualization **241**B of the token **222**B within the visualization **202**B of the data flow representation **240** may be visually emphasized as represented by arrow **245**B. Upon selecting control **243**C, a visualization **241**C of the token **222**C within the visualization **202**B of the data flow representation **240** may be visually emphasized as represented by arrow **245**C.

[0061] The visualization **202**B of the data flow representation **240** may also include other controls **251**A and **251**B that allow for other types of operations on the visualization **202**. For instance, as an example only, control **251**A might filter a view of the visualization to perhaps only a subset of the nodes of the data flow representation, or may perhaps aggregated nodes of the data flow representation. Control **252** might, for instance, visually emphasize visualized nodes having a particular relationship with a selected visualized node. For instance, the visually emphasized nodes might be the upstream and/or downstream nodes of a selected node of the data flow representation.

[0062] FIG. **3**A represents an example of a syntax tree **300**A and will be used as an example throughout the remainder of this description. However, the principles described herein apply regardless of the particular structure of the syntax tree **300** and the precise structure of the syntax tree **300** will depend on the content of the query script and the query language in which the query script is authored. In this particular syntax tree **300**, there are five nodes shown including nodes **310**A, **320**A, **330**A, **340**A and **350**A. Each node of the syntax tree **300**A is symbolically illustrated in FIG. **3**A as a circle. Furthermore, there are five relation **311**A, **321**A, **331**A, **341**A and **351**A. Each relation of the syntax tree is symbolically illustrated in FIG. **3**A as a dotted line.

[0063] FIG. **3**B represents an example of a data flow representation **300**B, and is similar to the syntax tree **300**A of FIG. **3**A. In the illustration of FIG. **3**B, each node of the data flow representation **300**B is represented as a square, and each flow is represented as an arrow line. In this example, there is one node of the data flow representation **300**B corresponding to each node of the syntax tree **300**A. For instance, nodes **310**A, **320**A, **330**A, **340**A and **350**A of the syntax tree **300**A correspond to respective nodes **310**B, **320**B, **330**B, **340**B and **350**B of the data flow representation **300**B. Furthermore, there is one data flow **311**B, **321**B, **331**B, **341**B and **351**B for each corresponding link **311**A, **321**A, **331**A, **341**A and **351**A of the syntax tree **300**A.

[0064] However, data flows often do not have one to one representations between links in the syntax diagram and data flows, and often there may be one or more nodes of a syntax tree in a single node of a data flow. Accordingly, the similarity in appearance between the syntax tree **300**A of FIG. **3**A and the data flow representation **300**B of FIG. **3**B is merely for purpose of clarity in explaining the principles described herein.

[0065] FIG. **4** illustrates a flowchart of a method **400** for correlating positions of query script to portions of a data flow representation of the query script. The method **400** might be performed just once with respect to a query script that is not anticipated to change. However, the method **400** might also be performed whenever a query script changes. Accordingly, the author of a query script might always be able to tell the correlation between the query script they are drafting and a data flow representation, and also be able to understand what portion of the query script corresponds to what portion of the data flow representation, and vice versa. The method **400** may be performed by, for instance, the computing system **100** of FIG. **1**.

[0066] The method **400** includes accessing a query script (act **401**). For instance, in FIG. **2**, the query script **210** is accessed. Again, this accessing might occur often, even perhaps whenever the query script changes in a small way. The query script is also visualized (act **410**). For instance, if the method **400** is performed by the computing system **100**, the query script might be displayed on a display, which is an example of the output mechanisms **112**A. In addition, a syntax tree is formulated in a manner that retains the script location marker of the tokens within the syntax tree (act **420**). Furthermore, the data flow representation is created from the syntax tree in a manner that retains the script location marker of the tokens within the data flow representation (act **421**). Accordingly, positions in the data flow representation are correlated with positions in the query script (act **422**) using the script location marker for at least some of the tokens included within the nodes in the syntax tree. Furthermore, the data flow representation is visualized (act **423**). For instance, if the method **400** is performed by the computing system **100**, the data flow representation might be displayed on a display, perhaps next to the visualization of the query script.

[0067] FIG. **5** illustrates a flowchart of an example method **500** for formulating the syntax tree and represents an example of the act **420** of FIG. **4**. The method **500** includes parsing the query script to generate multiple tokens (act **510**), each of at least some of the tokens having an associated script location marker that identifies a location in the query script from where the token originated. For instance, in Figure **2**, the query script **210** is parsed as represented by arrow **211** into tokens **220** having script location markers **223**A, **223**B and **223**C. This may be performed by, for instance, a parser of a compiler, the parser being slightly modified so that when a token is parsed out of the query script, a script location marker is created that identifies the query script location that the token came from, and then the script location marker is associated with the token such that when the token moves or is copied, the script location marker remains associated with the corresponding token.

[0068] Next, a syntax tree is formulated having multiple nodes (act **520**). As previously mentioned, each node of the syntax tree has one or more tokens parsed from the query script. The syntax tree retains the script location markers associated with the tokens, because the syntax tree includes the tokens and the script location markers follow the respective tokens. An example of a syntax tree is an Abstract Syntax Tree (AST).

[0069] FIG. **6** illustrates a flowchart of an example method **600** for formulating a data flow representation from a syntax tree and represents an example of the act **421** of FIG. **4**. The method **600** may be performed by the evaluator **235** of FIG. **2** for example, to build the data flow representation **240** from the syntax tree **230**. The method **600** includes first accessing (act **610**) the syntax tree. For instance, in FIG. **2**, the evaluator **235** access the syntax tree **230**. Again, an example of the syntax tree **230** is the syntax tree **300**A of FIG. **3**A.

[0070] The evaluator then evaluates (act **620**) each of at least some of the nodes of the syntax tree to identify the various data types of the node. For instance, the evaluator **635** of FIG. **2** evaluates each node of the syntax tree in order to identify the data types of input(s) and output(s) of the node. If the syntax tree **230** were structured as the syntax tree **300**A of FIG. **3**A, the evaluator would perform the act **620** for each of the nodes **310**A, **320**A, **330**A, **340**A and **350**A of the syntax tree. FIG. **7** illustrates a flowchart of a method **700** for evaluating a node of the syntax tree and represents one example of how the act **620** may be performed. FIG. **7** will be explained in detail further below.

[0071] The evaluator then formulates (act **630**) a data flow representation based on the syntax tree and augmented with the data types identified in the acts of evaluating. For instance, in FIG. **2**, the evaluator **235** formulates the data flow representation **240**.

[0072] As previously mentioned, in order to generate the data flow representation, the evaluator evaluates (act **620**) each of at least some of the nodes of the syntax tree. FIG. **7** illustrates a flowchart of a method **700** for evaluating a node of the syntax tree. The goal of the evaluation of each node is to identify a data type of any output(s) from that node.

[0073] First, the evaluator identifies (act **710**) a data type of one or more inputs to the node of the syntax tree. It may be that there are no inputs to the node of the syntax tree. In that case, act **710** may be skipped. Furthermore, it may be that due to upstream nodes not having been evaluated yet, the data type of one of the input(s) to the node may not yet be identifiable. In that case, the method **700** is deferred for that particular node of the syntax tree.

[0074] Accordingly, the evaluation of nodes is subject to evaluation of an order of dependency of the nodes of the syntax tree. For instance, referring to FIG. **3**A, node **310**A is evaluated so that the data types of the inputs **311**B to the node **320**A may be identified. Furthermore, node **320**A is evaluated prior to nodes **330**A and **340**A so that the inputs **321**B and **331**B to the respective nodes **330**A and **340**A may be identified. Nodes **230**A and **240**A are then evaluated so that inputs **341**B and **351**B to the node **350**A may be identified.

[0075] Once the input data type of the input(s) (if any) are determined for a given node of the syntax tree, the grammar set of the query script may then be applied to the one or more tokens of the node (act **720**) to thereby identify (act **730**) output data types of output(s), if any, of the node of the syntax tree.

[0076] The method **700** of FIG. **7** will now be described with respect to the syntax tree **300**A of FIG. **3**A. In order to generate the data flow representation **300**B of FIG. **3**B, the data types of each of the input(s), if any, and the output(s), if any, of the nodes of the syntax tree **300**A are determined. To do so, the method **700** is applied to each node of the syntax tree **300**A beginning at node **310**A, which is a dependee node for all other nodes of the syntax tree **300**A.

[0077] As for node **310**A, the data types of the input(s) of the node **310**A are identified (act **710**). In the case of node **310**A, there are no inputs to the node **310**A. The grammar rules of the query language are then applied (act **720**) to the token(s) of the node in order to identify (act **730**) an output data type **311**B of the node **310**A. By so doing, node **310**B having output data flows **311**B may be formulated (see FIG. **3**B). Node **320**A is then ready to be evaluated, being a dependent node from node **310**A, and given that the output data type of the output of its dependee node **310**A has been determined.

[0078] Again, the method **700** is performed, this time for node **320**A. As for node **320**A, the input(s) of the node **320**A are identified (act **710**). The input data type of the input of the node **320**A in this case is the same as the output type of the output **311**B of the node **310**B. Accordingly, the input data type can be readily identified. Now, the grammar rules of the query language are applied (act **720**) to the token(s) of the node **320**A in order to identify (act **730**) an output data type **321**B and **331**B of the node **320**A. By so doing, node **320**B having output data flows **321**B and **331**B may be formulated (see FIG. **3**B). Either and both of nodes **330**A and **340** are then ready to be evaluated.

[0079] When the method **700** is performed for node **330**A, the input(s) of the node **330**A are identified (act **710**). The input data type of the input of the node **330**A in this case is the same as the output data type of the output **321**B of the node **320**B. Accordingly, the input data type can be readily identified. Now, the grammar rules of the query language are applied (act **720**) to the token(s) of the node **330**A in order to identify (act **730**) an output data type **341**B of the node **330**A. By so doing, node **330**B having output data flow **341**B may be formulated (see FIG. **3**B).

[0080] When the method **700** is performed for node **340**A, the input(s) of the node **340**A are identified (act **710**). The input data type of the input of the node **340**A in this case is the same as the output data type of the output **331**B of the node **320**B. Accordingly, the input data type can be readily

identified. Now, the grammar rules of the query language are applied (act **720**) to the token(s) of the node **340**A in order to identify (act **730**) an output data type **351**B of the node **340**A. By so doing, node **340**B having output data flow **341**B may be formulated.

[0081] The method **700** may now be performed for node **350**A. The input types of inputs to the node **350**A are identified (act **710**). The input data types of the inputs of the node **350**A in this case is the same as the output data type of the output **341**B of the node **330**B, and the same as the output data type of the output **351**B of the node **340**B. There is no need to perform act **720** and **730** with respect to node **350**A since there are no output data flows from the node **350**A. Accordingly, node **350**B of the data flow representation **300**B may be formulated, thereby completing the formulation of the data flow representation **300**B of FIG. **3**B.

[0082] A user experience will now be described with respect to several user interfaces with respect to FIGS. **8** through **18**. Each of the FIGS. **8** through **18** represents a user interface that may be displayed on a display (e.g., one of the output mechanism **112**A) of the computing system (e.g., computing system **100**).

[0083] FIG. **8** illustrates a user interface in which only the query script is illustrated. Accordingly, the user interface of FIG. **8** represents an example of the visualization **201**B of FIG. **2**. To switch over to the visualization of the data flow representation, the user might select the "Diagram tab" represented within circle **810**. FIG. **9** illustrates the resulting user interface in this example user experience. Accordingly, FIG. **9** represents an example of the visualization **202**B of FIG. **2**.

[0084] Alternatively, in either FIG. **8** (the query script view) or FIG. **9** (the data flow representation view), the user might select the split control **820** to view the query script and the associated data flow representation side by side. FIG. **10** illustrates the resulting user interface showing the script view **1010** and the data flow representation view **1020**.

[0085] Selecting something in the data flow representation view **1020** will visually emphasize the relevant script. For instance, FIG. **11** is similar to FIG. **10**, except that the user has selected the node circled by circle **1110**. This results in the corresponding token being visually emphasized as represented by circle **1120**. Throughout FIGS. **11** through **18**, the selected node in the data flow representation view is represented by the node having rightward leaning hash marking. Otherwise emphasized nodes are represented by the node having dotted fill.

[0086] Conversely, selecting (e.g., putting the cursor over) somewhere in the script view will select the corresponding node in the data flow representation view. For instance, FIG. **12** is similar to FIG. **10**, except that the user has put the cursor at the location represented by circle **1210** in the script view, causing the corresponding node **1220** of the data flow representation view to be visually emphasized.

[0087] Note that in the views of FIGS. **11** and **12**, where a node of the data flow representation has been selected (either directly as in FIG. **11**, or view placing the cursor in the corresponding script portion as in FIG. **12**), the upstream and downstream nodes may be visually emphasized. This may be illustrated more clearly in FIG. **13**, in which node **1310** is selected. That causes the corresponding upstream nodes **1307**, **1308** and **1309**, and the corresponding downstream nodes **1311** and **1312** to be visually emphasized also.

[0088] The user might also choose to show only the related nodes of a given node by interacting with a control associated with that node. For instance, in FIG. **14**, the user might open a drop down menu **1410** and select the "Only related nodes" option **1420** resulting in the user interface of FIG. **15**. In FIG. **15**, only the selected node **1310**, and its related nodes **1307**, **1308**, **1309**, **1311** and **1312** are shown.

[0089] As illustrated in the user interface of FIG. **16**, there may be options for showing and hiding properties of the statement within the diagram (as represented within the circle **1610**). The user can choose how much of the properties, and which properties, to display. In this case, the user has selected to see the schema, the condition, the filter, and the sort properties of the nodes.

[0090] Furthermore, while FIGS. **9** through **16** show that the nodes of the data flow representation view represent operations (e.g., statements), and the edges represent data flows (e.g., rowsets), the user can switch this so that the nodes represent data flows and the edges represent operations. For instance, in FIG. **17**, the user may interact with a control, such as in the form of a drop down control **1710** and select the "Rowsets" option **1720**. FIG. **18** represents the resulting user interface with rowsets represented by nodes, and statements represented by edges.

[0091] Accordingly, an effective and automated mechanism for correlating query script positions with data flow representation positions has been described, along with various other convenient user experiences. This allows for more efficient drafting of correct and intended query script, and for the efficient evaluation of the same.

[0092] The present invention may be embodied in other specific forms without departing from its spirit or essential characteristics. The described embodiments are to be considered in all respects only as illustrative and not restrictive. The scope of the invention is, therefore, indicated by the appended claims rather than by the foregoing description. All changes which come within the meaning and range of equivalency of the claims are to be embraced within their scope.

What is claimed is:

1. A computing system comprising:

one or more processors;

one or more computer-readable storage media having thereon computer-executable instructions that are configured to be executed by the one or more processors to cause the computing system to configure the computing system to perform a method for correlating positions of query script to portions of a data flow representation of the query script, the method comprising:

an act of parsing the query script to generate a plurality of tokens, each of at least some of the plurality of tokens having an associated script location marker that identifies a location in the query script where the token originated;

an act of formulating a syntax tree having a plurality of nodes, each node comprising one or more of the tokens parsed from the query script;

an act formulating a data flow representation of the query script from the syntax tree; and

an act of correlating positions in the data flow representation with positions in the query script using a script location marker for at least some of the tokens included within the plurality of nodes in the syntax tree.

2. The computing system in accordance with claim 1, the method further comprising:

an act of visualizing the data flow representation.

3. The computing system in accordance with claim 2, the visualization of the data flow representation comprising a plurality of visualized nodes, each visualized node corresponding to one or more of the plurality of nodes of the syntax tree.

4. The computing system in accordance with claim 2, the method further comprising:

an act of visualizing the query script.

5. The computing system in accordance with claim 4, the visualization comprising a plurality of portions, each of the plurality of portions corresponding to a corresponding visualized portion of the query script based on a respective script location marker.

6. The computing system in accordance with claim 5, the visualization of the data flow representation having a plurality of controls for visually correlating visualized portions of the data flow representation with corresponding visualized portions of the query script.

7. The computing system in accordance with claim 2, the visualization of the data flow representation having a control for filtering a view of the visualization.

8. The computing system in accordance with claim 2, the visualization of the data flow representation having a control to visually emphasize visualized nodes having a particular relationship with a selected visualized node.

9. The computing system in accordance with claim 1, the associated script location marker of at least one of the tokens comprising a line identifier representing the line of the query script from which the token was taken.

10. The computing system in accordance with claim 1, the act of formulating a data flow representation of the query script from the syntax tree comprising:

an act of evaluating each of at least some of the plurality of nodes of the syntax tree to do the following for each evaluated node: an act of identifying a data type of one or more inputs to the node; an act of applying a grammar set of a query language of the query script to the one or more tokens of the node to identify an output data type of one or more outputs from the node; and an act of formulating a data flow representation based on the syntax tree and augmented with the data types identified in the acts of evaluating.

11. The computing system in accordance with claim 10, the query language being a big data query language.

12. A method for correlating positions of query script to portions of a data flow representation of the query script, the method comprising:

an act of parsing the query script to generate a plurality of tokens, each of at least some of the plurality of tokens having an associated script location marker that identifies a location in the query script where the token originated;

an act of formulating a syntax tree having a plurality of nodes, each node comprising one or more of the tokens parsed from the query script;

an act formulating a data flow representation of the query script from the syntax tree; and

an act of correlating positions in the data flow representation with positions in the query script using a script location marker for at least some of the tokens included within the plurality of nodes in the syntax tree.

13. The method in accordance with claim 12, the method further comprising:

an act of visualizing the data flow representation.

14. The method in accordance with claim 13, the visualization of the data flow representation comprising a plurality of visualized nodes, each visualized node corresponding to one or more of the plurality of nodes of the syntax tree.

15. The method in accordance with claim 13, the method further comprising:

an act of visualizing the query script.

16. The method in accordance with claim 15, the visualization comprising a plurality of portions, each of the plurality of portions corresponding to a corresponding visualized portion of the script location marker.

17. The method in accordance with claim 16, the visualization of the data flow representation having a plurality of controls for visually correlating visualized portions of the data flow representation with corresponding visualized portions of the query script.

18. The method in accordance with claim 13, the visualization of the data flow representation having a control for filtering a view of the visualization.

19. The method in accordance with claim 13, the visualization of the data flow representation having a control to visually emphasize visualized nodes having a particular relationship with a selected visualized node.

20. A computer program product comprising one or more computer-readable storage media having thereon computer-executable instructions that are structured such that, when executed by one or more processors of the computing system, cause the computing system to perform a method for correlating positions of query script to portions of a data flow representation of the query script, the method comprising:

an act of parsing the query script to generate a plurality of tokens, each of at least some of the plurality of tokens having an associated script location marker that identifies a location in the script where the token originated;

an act of formulating a syntax tree having a plurality of nodes, each node comprising one or more of the tokens parsed from the query script;

an act formulating a data flow representation of the query script from the syntax tree; and

an act of correlating positions in the data flow representation with positions in the query script using a script location marker for at least some of the tokens included within the plurality of nodes in the syntax tree.

* * * * *