(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
17 November 2016 (17.11.2016)

WIPO | PCT

(10) International Publication Number
WO 2016/183547 A1

(51) International Patent Classification:
G06F 7/00 (2006.01) *G06F 17/30* (2006.01)

(21) International Application Number:
PCT/US2016/032589

(22) International Filing Date:
14 May 2016 (14.05.2016)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
62/161,813 14 May 2015 (14.05.2015) US

(71) Applicant: **WALLEYE SOFTWARE, LLC** [US/US]; 2800 Niagara Lane N., Plymouth, MN 55447 (US).

(72) Inventors: **TEODORESCU, Radu**; 303 East 109th Street, Apt 1, New York, NY 10029 (US). **CAUDY, Ryan**; 360 East 88th Street, Apt 25b, New York, NY 10128 (US). **KENT, David, R.**; 6965 Winter Hawk Circle, Colorado Springs, CO 80919 (US). **WRIGHT, Charles**; 48 Furnace Woods Road, Cortland Manor, NY 10567 (US). **ZELDIS, Mark**; 16 Castle Court, Randolph, NJ 07869 (US).

(74) Agent: **CARMICHAEL, James, T.**; Carmichael Ip, Pllc, 8000 Towres Crescent Drive, 13th Floor, Tysons Corner, VA 22182 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

[Continued on next page]

(54) Title: PARSING AND COMPILING DATA SYSTEM QUERIES

(57) Abstract: Described are methods, systems and computer readable media for parsing and compiling data system queries.

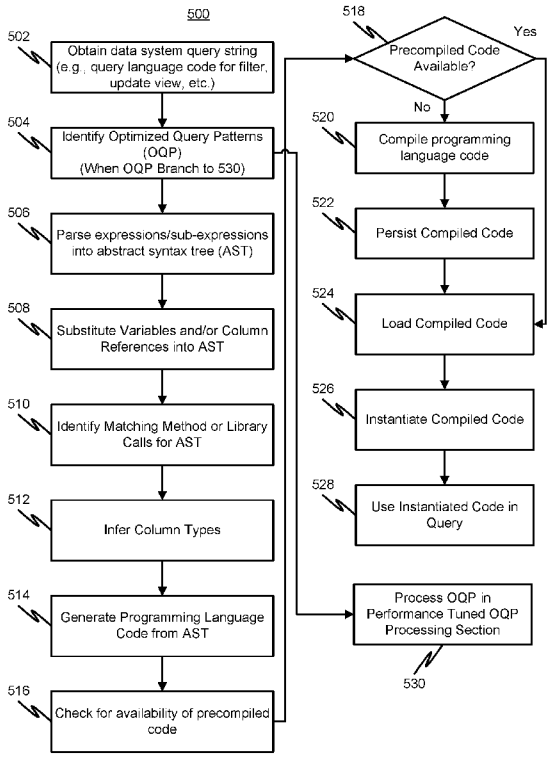

FIG. 5

WO 2016/183547 A1

(84) **Designated States** *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report (Art. 21(3))*

PARSING AND COMPILING DATA SYSTEM QUERIES

[0001] This application claims the benefit of U.S. Provisional Application No. 62/161,813, entitled "Computer Data System" and filed on May 14, 2015, which is incorporated herein by reference in its entirety.

[0002] Embodiments relate generally to computer data systems, and more particularly, to methods, systems and computer readable media for parsing and compiling data system queries.

[0003] Some conventional computer data systems may provide a query language in which a query is interpreted by the computer data system to produce a query results. These query languages may have a static grammar with a fixed number of commands or operators. These conventional query languages may not be extensible and may not provide for operations outside of the static grammar. A need may exist to provide a data system parser and compiler that can parse and compile a data system query written in a query language that permits inclusion of programming language code or constructs, where a result of the parsing and compiling is compiled programming language code suitable for execution on a processor. Further, a need may exist to provide a concise, expressive data system query language. Also, a need may exist to provide an expressive data system query language along with improved data system query execution performance.

[0004] Some implementations were conceived in light of the above mentioned needs, problems and/or limitations, among other things.

[0005] Some implementations can include a system for parsing, generating code and compiling computer data system query language code, the system comprising one or more hardware processors coupled to a nontransitory computer readable medium having stored thereon software instructions that, when executed by the one or more processors, cause the one or more processors to perform operations. The operations can include obtaining, at the one or more hardware processors, computer data system query language code from an electronic computer readable data storage, and parsing, at the one or more hardware processors, the computer data system query language code to generate a computer data system query language parsed code structure.

[0006] The operations can also include resolving, at the one or more hardware processors, a type of one or more columns represented in the parsed code structure, and inserting, at the one or

more hardware processors, resolved types into the parsed code structure. The operations can further include generating, at the one or more hardware processors, computer programming language code from the computer data system query language parsed code structure, and determining, at the one or more hardware processors, whether precompiled code corresponding to the generated computer programming language code is available in a precompiled code repository stored in the electronic computer readable data storage.

[0007] The operations can also include, when precompiled code is available in the precompiled code repository, loading, at the one or more hardware processors, the precompiled code. The operations can further include, when precompiled code is not available in the precompiled code repository, compiling, at the one or more hardware processors, the computer programming language code to generate compiled computer programming language code, and loading, at the one or more hardware processors, the compiled computer programming language code.

[0008] The operations can also include instantiating, at the one or more hardware processors, the loaded precompiled code or the compiled computer programming language code, and executing, at the one or more hardware processors, the instantiated code to perform a query operation corresponding to the computer data system query language code.

[0009] The computer data system query language code can include one or more instructions of a data system query language. The operations can further include identifying a source of columns or query scope variables for substitution and use in computer programming language code. The operations can further include persisting the compiled computer programming language code by storing the precompiled code in the precompiled code repository.

[0010] Determining whether precompiled code corresponding to the programming language code is available in a precompiled code repository can include generating a token representing the generated computer programming language code, wherein the token includes a result of a hash function of one or more attributes of the computer programming language code, comparing the token to one or more repository tokens in the precompiled code repository corresponding to precompiled code units, and based on the comparing, determining whether the token matches any of the repository tokens.

[0011] The inserting can include inserting references to data objects and variables that have been made available to the query language code. The operations can further include repeating the resolving and inserting until any unresolved columns or variables have been resolved.

[0012] Some implementations can include a method for parsing, generating code and compiling computer data system query language code. The method can include obtaining, at a hardware processor, computer data system query language code from an electronic computer readable data storage, and parsing, at the hardware processor, the computer data system query language code to generate a computer data system query language parsed code structure. The method can also include resolving, at the hardware processor, a type of one or more columns represented in the parsed code structure, and inserting, at the hardware processor, resolved types into the parsed code structure. The method can further include generating, at the hardware processor, computer programming language code from the computer data system query language parsed code structure, and determining, at the hardware processor, whether precompiled code corresponding to the generated computer programming language code is available in a precompiled code repository stored in the electronic computer readable data storage.

[0013] The method can also include when precompiled code is available in the precompiled code repository, loading, at the hardware processor, the precompiled code. The method can further include, when precompiled code is not available in the precompiled code repository, compiling, at the hardware processor, the computer programming language code to generate compiled computer programming language code, and loading, at the hardware processor, the compiled computer programming language code.

[0014] The method can also include instantiating, at the hardware processor, the loaded precompiled code or the compiled computer programming language code, and executing, at the hardware processor, the instantiated code to perform a query operation corresponding to the computer data system query language code. The computer data system query language code includes one or more instructions of a data system query language.

[0015] The method can also include identifying a source of columns or query scope variables for substitution and use in computer programming language code. The method can further include persisting the compiled computer programming language code by storing the precompiled code in the precompiled code repository.

[0016] Determining whether precompiled code corresponding to the programming language code is available in a precompiled code repository comprises generating a token representing the generated computer programming language code, wherein the token includes a result of a hash function of one or more attributes of the computer programming language code, comparing the

token to one or more repository tokens in the precompiled code repository corresponding to precompiled code units, and based on the comparing, determining whether the token matches any of the repository tokens.

[0017] The inserting can include inserting references to data objects and variables that have been made available to the query language code. The method can further include repeating the resolving and inserting until any unresolved columns or variables have been resolved.

[0018] Some implementations can include a nontransitory computer readable medium having stored thereon software instructions that, when executed by one or more processors, cause the one or more processors to perform operations. The operations can include obtaining, at the one or more hardware processors, computer data system query language code from an electronic computer readable data storage, and parsing, at the one or more hardware processors, the computer data system query language code to generate a computer data system query language parsed code structure.

[0019] The operations can also include resolving, at the one or more hardware processors, a type of one or more columns represented in the parsed code structure, and inserting, at the one or more hardware processors, resolved types into the parsed code structure. The operations can further include generating, at the one or more hardware processors, computer programming language code from the computer data system query language parsed code structure, and determining, at the one or more hardware processors, whether precompiled code corresponding to the generated computer programming language code is available in a precompiled code repository stored in the electronic computer readable data storage.

[0020] The operations can also include, when precompiled code is available in the precompiled code repository, loading, at the one or more hardware processors, the precompiled code. The operations can further include, when precompiled code is not available in the precompiled code repository, compiling, at the one or more hardware processors, the computer programming language code to generate compiled computer programming language code, and loading, at the one or more hardware processors, the compiled computer programming language code.

[0021] The operations can also include instantiating, at the one or more hardware processors, the loaded precompiled code or the compiled computer programming language code, and executing, at the one or more hardware processors, the instantiated code to perform a query operation corresponding to the computer data system query language code.

[0022] The computer data system query language code can include one or more instructions of a data system query language. The operations can further include identifying a source of columns or query scope variables for substitution and use in computer programming language code. The operations can further include persisting the compiled computer programming language code by storing the precompiled code in the precompiled code repository.

[0023] Determining whether precompiled code corresponding to the programming language code is available in a precompiled code repository can include generating a token representing the generated computer programming language code, wherein the token includes a result of a hash function of one or more attributes of the computer programming language code, comparing the token to one or more repository tokens in the precompiled code repository corresponding to precompiled code units, and based on the comparing, determining whether the token matches any of the repository tokens.

[0024] The inserting can include inserting references to data objects and variables that have been made available to the query language code. The operations can further include repeating the resolving and inserting until any unresolved columns or variables have been resolved.

BRIEF DESCRIPTION OF THE DRAWINGS

[0025] FIG. 1 is a diagram of an example computer data system showing an example data distribution configuration in accordance with some implementations.

[0026] FIG. 2 is a diagram of an example computer data system showing an example administration/process control arrangement in accordance with some implementations.

[0027] FIG. 3 is a diagram of an example computing device configured for parsing and compiling data system queries in accordance with some implementations.

[0028] FIG. 4 is a diagram showing a data system parser and compiler in accordance with some implementations.

[0029] FIG. 5 is a flowchart showing an example method for parsing and compiling data system queries in accordance with some implementations.

DETAILED DESCRIPTION

[0030] Reference may be made herein to the Java programming language, Java classes, Java bytecode and the Java Virtual Machine (JVM) for purposes of illustrating example implementations. It will be appreciated that implementations can include other programming languages (e.g., groovy, Scala, R, Go, etc.), other programming language structures as an alternative to or in addition to Java classes (e.g., other language classes, objects, data structures, program units, code portions, script portions, etc.), other types of bytecode, object code and/or executable code, and/or other virtual machines or hardware implemented machines configured to execute a data system query.

[0031] FIG. 1 is a diagram of an example computer data system and network 100 showing an example data distribution configuration in accordance with some implementations. In particular, the system 100 includes an application host 102, a periodic data import host 104, a query server host 106, a long-term file server 108, and a user data import host 110. While tables are used as an example data object in the description below, it will be appreciated that the data system described herein can also process other data objects such as mathematical objects (e.g., a singular value decomposition of values in a given range of one or more rows and columns of a table), TableMap objects, etc. A TableMap object provides the ability to lookup a Table by some key. This key represents a unique value (or unique tuple of values) from the columns aggregated on in a byExternal() statement execution, for example. A TableMap object can be the result of a byExternal() statement executed as part of a query. It will also be appreciated that the configurations shown in FIGS. 1 and 2 are for illustration purposes and in a given implementation each data pool (or data store) may be directly attached or may be managed by a file server.

[0032] The application host 102 can include one or more application processes 112, one or more log files 114 (e.g., sequential, row-oriented log files), one or more data log tailers 116 and a multicast key-value publisher 118. The periodic data import host 104 can include a local table data server, direct or remote connection to a periodic table data store 122 (e.g., a column-oriented table data store) and a data import server 120. The query server host 106 can include a multicast key-value subscriber 126, a performance table logger 128, local table data store 130 and one or more remote query processors (132, 134) each accessing one or more respective

tables (136, 138). The long-term file server 108 can include a long-term data store 140. The user data import host 110 can include a remote user table server 142 and a user table data store 144. Row-oriented log files and column-oriented table data stores are discussed herein for illustration purposes and are not intended to be limiting. It will be appreciated that log files and/or data stores may be configured in other ways. In general, any data stores discussed herein could be configured in a manner suitable for a contemplated implementation.

[0033] In operation, the input data application process 112 can be configured to receive input data from a source (e.g., a securities trading data source), apply schema-specified, generated code to format the logged data as it's being prepared for output to the log file 114 and store the received data in the sequential, row-oriented log file 114 via an optional data logging process. In some implementations, the data logging process can include a daemon, or background process task, that is configured to log raw input data received from the application process 112 to the sequential, row-oriented log files on disk and/or a shared memory queue (e.g., for sending data to the multicast publisher 118). Logging raw input data to log files can additionally serve to provide a backup copy of data that can be used in the event that downstream processing of the input data is halted or interrupted or otherwise becomes unreliable.

[0034] A data log tailer 116 can be configured to access the sequential, row-oriented log file(s) 114 to retrieve input data logged by the data logging process. In some implementations, the data log tailer 116 can be configured to perform strict byte reading and transmission (e.g., to the data import server 120). The data import server 120 can be configured to store the input data into one or more corresponding data stores such as the periodic table data store 122 in a column-oriented configuration. The periodic table data store 122 can be used to store data that is being received within a time period (e.g., a minute, an hour, a day, etc.) and which may be later processed and stored in a data store of the long-term file server 108. For example, the periodic table data store 122 can include a plurality of data servers configured to store periodic securities trading data according to one or more characteristics of the data (e.g., a data value such as security symbol, the data source such as a given trading exchange, etc.).

[0035] The data import server 120 can be configured to receive and store data into the periodic table data store 122 in such a way as to provide a consistent data presentation to other parts of the system. Providing/ensuring consistent data in this context can include, for example, recording logged data to a disk or memory, ensuring rows presented externally are available for

consistent reading (e.g., to help ensure that if the system has part of a record, the system has all of the record without any errors), and preserving the order of records from a given data source. If data is presented to clients, such as a remote query processor (132, 134), then the data may be persisted in some fashion (e.g., written to disk).

[0036] The local table data server 124 can be configured to retrieve data stored in the periodic table data store 122 and provide the retrieved data to one or more remote query processors (132, 134) via an optional proxy.

[0037] The remote user table server (RUTS) 142 can include a centralized consistent data writer, as well as a data server that provides processors with consistent access to the data that it is responsible for managing. For example, users can provide input to the system by writing table data that is then consumed by query processors.

[0038] The remote query processors (132, 134) can use data from the data import server 120, local table data server 124 and/or from the long-term file server 108 to perform queries. The remote query processors (132, 134) can also receive data from the multicast key-value subscriber 126, which receives data from the multicast key-value publisher 118 in the application host 102. The performance table logger 128 can log performance information about each remote query processor and its respective queries into a local table data store 130. Further, the remote query processors can also read data from the RUTS, from local table data written by the performance logger, or from user table data read over NFS, for example.

[0039] It will be appreciated that the configuration shown in FIG. 1 is a typical example configuration that may be somewhat idealized for illustration purposes. An actual configuration may include one or more of each server and/or host type. The hosts/servers shown in FIG. 1 (e.g., 102-110, 120, 124 and 142) may each be separate or two or more servers may be combined into one or more combined server systems. Data stores can include local/remote, shared/isolated and/or redundant. Any table data may flow through optional proxies indicated by an asterisk on certain connections to the remote query processors. Also, it will be appreciated that the term "periodic" is being used for illustration purposes and can include, but is not limited to, data that has been received within a given time period (e.g., millisecond, second, minute, hour, day, week, month, year, etc.) and which has not yet been stored to a long-term data store (e.g., 140).

[0040] FIG. 2 is a diagram of an example computer data system 200 showing an example administration/process control arrangement in accordance with some implementations. The

system 200 includes a production client host 202, a controller host 204, a GUI host or workstation 206, and query server hosts 208 and 210. It will be appreciated that there may be one or more of each of 202-210 in a given implementation.

[0041] The production client host 202 can include a batch query application 212 (e.g., a query that is executed from a command line interface or the like) and a real time query data consumer process 214 (e.g., an application that connects to and listens to tables created from the execution of a separate query). The batch query application 212 and the real time query data consumer 214 can connect to a remote query dispatcher 222 and one or more remote query processors (224, 226) within the query server host 1 208.

[0042] The controller host 204 can include a persistent query controller 216 configured to connect to a remote query dispatcher 232 and one or more remote query processors 228-230. In some implementations, the persistent query controller 216 can serve as the "primary client" for persistent queries and can request remote query processors from dispatchers, and send instructions to start persistent queries. For example, a user can submit a query to the persistent query controller 216, and the persistent query controller 216 starts and runs the query every day. In another example, a securities trading strategy could be a persistent query. The persistent query controller can start the trading strategy query every morning before the market opened, for instance. It will be appreciated that 216 can work on times other than days. In some implementations, the controller may require its own clients to request that queries be started, stopped, etc. This can be done manually, or by scheduled (e.g., cron jobs). Some implementations can include "advanced scheduling" (e.g., auto-start/stop/restart, time-based repeat, etc.) within the controller.

[0043] The GUI/host workstation can include a user console 218 and a user query application 220. The user console 218 can be configured to connect to the persistent query controller 216. The user query application 220 can be configured to connect to one or more remote query dispatchers (e.g., 232) and one or more remote query processors (228, 230).

[0044] FIG. 3 is a diagram of an example computing device 300 in accordance with at least one implementation. The computing device 300 includes one or more processors 302, operating system 304, computer readable medium 306 and network interface 308. The memory 306 can include a data system query parser and compiler application 310 and a data section 312 (e.g., for storing data system query strings, abstract syntax trees, precompiled code, etc.).

[0045] In operation, the processor 302 may execute the application 310 stored in the memory 306. The application 310 can include software instructions that, when executed by the processor, cause the processor to perform operations for parsing and compiling data system queries in accordance with the present disclosure (e.g., performing one or more of 502-528 described below). The application program 310 can operate in conjunction with the data section 312 and the operating system 304.

[0046] FIG. 4 shows an example parser and compiler configuration 400 for use with a data system. The configuration 400 includes a parser/code generator 404, a compiler 408 and a precompiled code repository 412.

[0047] In operation, a data system query language string 402 is provided to the parser/code generator 404. The data system query language string 402 can include one or more of: a data system query language string, an object oriented programming language code string (e.g., Java code, Groovy code, etc.), other programming language string (e.g., R programming language code), or the like. Also, a data system query language string 402 may be augmented by code generated by a code generator, thus helping the data system query language to be concise.

[0048] The data system query language string 402 can be parsed by the parser/code generator 404 into computer language code 406. The parser/code generator 404 may be configured to parse computer data system language code, and then produce code in another computer programming language. As part of the two-phase parsing/code generation operation, the parser may generate an abstract syntax tree (AST), which can be used by the code generator to generate computer language code and to infer the type of any data columns and/or tables produced by the data system query language string. For example, a string in a computer data system language can be parsed into an AST that is then used by the code generator to generate Java code having properly inferred types. The parser/code generator 404 can also vectorize operations from the query language code. For example, for a query language statement of "a=b+c", where b and c are column sources, the code generator 404 can generate a looping code structure to perform the operation, for example "ai = bi + ci, for all i values in the column sources."

[0049] The computer language code 406 is provided as input to the compiler 408. Based on one or more attributes and/or derived attributes of the computer language code 406, the code generator or compiler can determine whether there is a precompiled class (or other precompiled code) for the computer language code 406. The attributes and/or derived attributes can include a

hash function result value of one or more attributes of the computer language code 406 such as class file name, object name, one or more parameters, a portion of the code itself, etc.

[0050] When the precompiled code repository 412 contains precompiled code (e.g., one or more precompiled Java class files) that corresponds to the computer language code 406, the precompiled code can be used, which permits the system to avoid using processing time to compile the computer language code 406. If precompiled code corresponding to the computer language code 406 is not found in the repository, the compiler compiles the computer language code 406 to generate compiled programming language code (e.g., one or more compiled Java class files) and optionally add the compiled code to the precompiled code repository for future reuse. The precompiled code library can be updated over time to include compiled code not found in the repository during a parsing/compiling process (or be updated to remove compiled code). Further details of the parsing and compiling are described below in connection with FIG. 5.

[0051] FIG. 5 a flowchart showing an example method 500 for parsing and compiling data system queries in accordance with some implementations. Processing begins at 502, where a data system query language string is obtained. The query language string (e.g., 402) can be obtained from another system, from a file sent by a user, from a command line interface or the like. The query language string can include one or more data system query operations including but not limited to filtering, updating and/or viewing data retrieved from the data system or created by a query. Processing continues to 504.

[0052] At 504, the parser identifies whether the query string is an optimized query pattern (OQP), which can include, for example, special, very common queries that have been performance tuned.

[0053] Some simple examples include:

[0054] "Symbol in `AAPL`,`GOOG`"

[0055] "A = 13"

[0056] "B < 12"

[0057] If the query string is identified as an OQP, processing continues to 530, where the OQP is processed via a special performance tuned OQP processing section without requiring code generation / compilation, etc. Otherwise processing continues to 506.

[0058] At 506, one or more expressions (or subexpressions) within the query string are parsed into a syntax tree (e.g., an abstract syntax tree or AST). The AST can be used to provide contextual information to the compiler in later stages described below. The AST can include a tree representation of the abstract syntactic structure of source code written in a programming language (e.g., the query string). Each node of the AST can represent a construct in the source code. Processing continues to 508.

[0059] At 508, variables and/or column representations are substituted into the AST. Because the query language string may include references to data within a row, column or table of the data system, the parser may need to substitute the programming language representation of certain variables, column names, table names etc. with representations that are suitable for the compilation process. For example, assume the following code:

[0060] a = 13

[0061] t2 = t1.update("X=A+a")

[0062] In this example, there is a variable "a" and a column "A". When "X=A+a" goes through the code generation and compilation process, the system recognizes that "A" is a column and "a" is a variable that was defined in a scope outside of the snippet we are compiling. Processing continues to 510.

[0063] At 510, matching method or library calls are identified within the AST. For example, assume a code string of t2 = t1.update("X=func(A,2)"). When the string "X=func(A,2)" is parsed and compiled, the system needs to determine what "func" is. Here, the system can determine that 2 is an "int". From the type of column A, know the type of A -- let's say "float" for this example. Now the system needs to find an appropriate function for "func(float,int)". If we are able to find an exact match, we use it. We may have to handle type conversions. For example "func(double,long)" may be the closest match. In general, the system is performing this step to determine what the correct function is. Processing continues to 512.

[0064] At 512, column types within the AST are inferred. Because a query string can create one or more tables having one or more columns each, the compiler may need to have type information for the columns created by the query string. Often, determining variable or object type within a programming language can be difficult, especially for data objects or structures created dynamically from a language such as the data system query language described herein in which the user may not be required to declare a type of a data column. Without a type

declaration, a compiler may have to resort to using a lowest common denominator type or catch all type (e.g., java.lang.Object in Java) as a substitute for the actual type of a column created by the query string.

[0065] To infer (or resolve) the type of a column, the parser traverses the AST in order to determine a context of the column in question. The context of the column in question can include the type of variables or objects related to the column within the AST (e.g., return types, argument types, etc.). The parser can evaluate the type of the adjacent variables or objects to infer (or resolve) the type of the column in question. The resolution of the type can follow standard conventions once the context of the column in question has been determined. For example, if a column having an unknown type is defined to contain the result of a hypothetical mathematical operator "plus" and the parameters to the "plus" operator are both of type "int" or integer, the parser may identify a "plus" function that takes two variables of type "int" as parameters and returns a value of type "double" as a result. Because the parser has identified the plus operator that matches the input parameter context, the return type of "double" from the "plus" function can be used to resolve the type of the column to double and a column with a correct type can be created to hold the results of the "plus" function. The parser continues traversing the AST until all unknown column types are resolved. Processing continues to 514.

[0066] At 514, the AST with unknown columns types resolved is translated into (or used to generate) programming language code. For example, the AST may be used by the code generator to generate Java language code. The code generation can also include adding programming language boilerplate for compilation purposes, providing information to permit access to relevant variables within scope, and adding information to permit access to relevant libraries and/or classes which may be in scope for the query. Processing continues to 516.

[0067] At 516, once the programming language code is available, the parser/compiler system can determine whether precompiled code corresponding to the translated code is available. The parser/compiler system can use one or more attributes of the translated code to generate a token for comparison to precompiled code sections within a precompiled code repository (e.g., 412). The token can include a result of processing one or more attributes of the translated code using a hash function. The attributes can include one or more of the code class name, code file name, a portion of the code, or the like. Processing continues to 518.

[0068] At 518, the result of the determining whether a precompiled version of the translated code is available is evaluated. If precompiled code is present, processing continues to 524. Otherwise, processing continues to 520.

[0069] At 520, the translated code is compiled by a compiler (e.g., 408) into compiled programming language code (e.g., 410). For example, the translated code may be Java language code that is compiled into one or more Java language classes. Processing continues to 522.

[0070] At 522, the compiled code is persisted (or stored) in a precompiled code repository (e.g., 412) along with one or more tokens (e.g., a result of hash function) that can be used to identify and retrieve the precompiled code. Processing continues to 524.

[0071] At 524, the compiled code is loaded. The compiled code may be the newly compiled code resulting from 520 or precompiled code identified at 516/518. Processing continues to 526.

[0072] At 526, the loaded code is instantiated (e.g., prepared for use, constructed in memory for execution, or the like). Processing continues to 528.

[0073] At 528, the instantiated code is executed to perform the query function specified in the query string provided at 502. It will be appreciated that 502-528 may be repeated in whole or in part in order to accomplish a contemplated query task.

[0074] It will be appreciated that the modules, processes, systems, and sections described above can be implemented in hardware, hardware programmed by software, software instructions stored on a nontransitory computer readable medium or a combination of the above. A system as described above, for example, can include a processor configured to execute a sequence of programmed instructions stored on a nontransitory computer readable medium. For example, the processor can include, but not be limited to, a personal computer or workstation or other such computing system that includes a processor, microprocessor, microcontroller device, or is comprised of control logic including integrated circuits such as, for example, an Application Specific Integrated Circuit (ASIC), a field programmable gate array (FPGA), a graphics processing unit (GPU), or the like. The instructions can be compiled from source code instructions provided in accordance with a programming language such as Java, C, C++, C#.net, assembly or the like. The instructions can also comprise code and data objects provided in accordance with, for example, the Visual Basic™ language, a specialized database query language, or another structured or object-oriented programming language. The sequence of programmed instructions, or programmable logic device configuration software, and data

associated therewith can be stored in a nontransitory computer-readable medium such as a computer memory or storage device which may be any suitable memory apparatus, such as, but not limited to ROM, PROM, EEPROM, RAM, flash memory, disk drive and the like.

[0075] Furthermore, the modules, processes systems, and sections can be implemented as a single processor or as a distributed processor. Further, it should be appreciated that the steps mentioned above may be performed on a single or distributed processor (single and/or multi-core, or cloud computing system). Also, the processes, system components, modules, and sub-modules described in the various figures of and for embodiments above may be distributed across multiple computers or systems or may be co-located in a single processor or system. Example structural embodiment alternatives suitable for implementing the modules, sections, systems, means, or processes described herein are provided below.

[0076] The modules, processors or systems described above can be implemented as a programmed general purpose computer, an electronic device programmed with microcode, a hard-wired analog logic circuit, software stored on a computer-readable medium or signal, an optical computing device, a networked system of electronic and/or optical devices, a special purpose computing device, an integrated circuit device, a semiconductor chip, and/or a software module or object stored on a computer-readable medium or signal, for example.

[0077] Embodiments of the method and system (or their sub-components or modules), may be implemented on a general-purpose computer, a special-purpose computer, a programmed microprocessor or microcontroller and peripheral integrated circuit element, an ASIC or other integrated circuit, a digital signal processor, a hardwired electronic or logic circuit such as a discrete element circuit, a programmed logic circuit such as a PLD, PLA, FPGA, PAL, or the like. In general, any processor capable of implementing the functions or steps described herein can be used to implement embodiments of the method, system, or a computer program product (software program stored on a nontransitory computer readable medium).

[0078] Furthermore, embodiments of the disclosed method, system, and computer program product (or software instructions stored on a nontransitory computer readable medium) may be readily implemented, fully or partially, in software using, for example, object or object-oriented software development environments that provide portable source code that can be used on a variety of computer platforms. Alternatively, embodiments of the disclosed method, system, and computer program product can be implemented partially or fully in hardware using, for example,

standard logic circuits or a VLSI design. Other hardware or software can be used to implement embodiments depending on the speed and/or efficiency requirements of the systems, the particular function, and/or particular software or hardware system, microprocessor, or microcomputer being utilized. Embodiments of the method, system, and computer program product can be implemented in hardware and/or software using any known or later developed systems or structures, devices and/or software by those of ordinary skill in the applicable art from the function description provided herein and with a general basic knowledge of the software engineering and computer networking arts.

[0079] Moreover, embodiments of the disclosed method, system, and computer readable media (or computer program product) can be implemented in software executed on a programmed general purpose computer, a special purpose computer, a microprocessor, or the like.

[0080] It is, therefore, apparent that there is provided, in accordance with the various embodiments disclosed herein, methods, systems and computer readable media for parsing and compiling data system queries.

[0081] Application No. _____, entitled "DATA PARTITIONING AND ORDERING" (Attorney Docket No. W1.1-10057) and filed in the United States Patent and Trademark Office on May 14, 2016, is hereby incorporated by reference herein in its entirety as if fully set forth herein.

[0082] Application No. _____, entitled "COMPUTER DATA SYSTEM DATA SOURCE REFRESHING USING AN UPDATE PROPAGATION GRAPH" (Attorney Docket No. W1.4-10058) and filed in the United States Patent and Trademark Office on May 14, 2016, is hereby incorporated by reference herein in its entirety as if fully set forth herein.

[0083] Application No. _____, entitled "COMPUTER DATA SYSTEM POSITION-INDEX MAPPING" (Attorney Docket No. W1.5-10083) and filed in the United States Patent and Trademark Office on May 14, 2016, is hereby incorporated by reference herein in its entirety as if fully set forth herein.

[0084] Application No. _____, entitled "SYSTEM PERFORMANCE LOGGING OF COMPLEX REMOTE QUERY PROCESSOR QUERY OPERATIONS" (Attorney Docket No. W1.6-10074) and filed in the United States Patent and Trademark Office on May 14, 2016, is hereby incorporated by reference herein in its entirety as if fully set forth herein.

[0085] Application No. _____, entitled "DISTRIBUTED AND OPTIMIZED GARBAGE COLLECTION OF REMOTE AND EXPORTED TABLE HANDLE LINKS TO UPDATE PROPAGATION GRAPH NODES" (Attorney Docket No. W1.8-10085) and filed in the United States Patent and Trademark Office on May 14, 2016, is hereby incorporated by reference herein in its entirety as if fully set forth herein.

[0086] Application No. _____, entitled "COMPUTER DATA SYSTEM CURRENT ROW POSITION QUERY LANGUAGE CONSTRUCT AND ARRAY PROCESSING QUERY LANGUAGE CONSTRUCTS" (Attorney Docket No. W2.1-10060) and filed in the United States Patent and Trademark Office on May 14, 2016, is hereby incorporated by reference herein in its entirety as if fully set forth herein.

[0087] Application No. _____, entitled "PARSING AND COMPILING DATA SYSTEM QUERIES" (Attorney Docket No. W2.2-10062) and filed in the United States Patent and Trademark Office on May 14, 2016, is hereby incorporated by reference herein in its entirety as if fully set forth herein.

[0088] Application No. _____, entitled "DYNAMIC FILTER PROCESSING" (Attorney Docket No. W2.4-10075) and filed in the United States Patent and Trademark Office on May 14, 2016, is hereby incorporated by reference herein in its entirety as if fully set forth herein.

[0089] Application No. _____, entitled "DYNAMIC JOIN PROCESSING USING REAL-TIME MERGED NOTIFICATION LISTENER" (Attorney Docket No. W2.6-10076) and filed in the United States Patent and Trademark Office on May 14, 2016, is hereby incorporated by reference herein in its entirety as if fully set forth herein.

[0090] Application No. _____, entitled "DYNAMIC TABLE INDEX MAPPING" (Attorney Docket No. W2.7-10077) and filed in the United States Patent and Trademark Office on May 14, 2016, is hereby incorporated by reference herein in its entirety as if fully set forth herein.

[0091] Application No. _____, entitled "QUERY TASK PROCESSING BASED ON MEMORY ALLOCATION AND PERFORMANCE CRITERIA" (Attorney Docket No. W2.8-10094) and filed in the United States Patent and Trademark Office on May 14, 2016, is hereby incorporated by reference herein in its entirety as if fully set forth herein.

[0092] Application No. _____, entitled "A MEMORY-EFFICIENT COMPUTER SYSTEM FOR DYNAMIC UPDATING OF JOIN PROCESSING" (Attorney Docket No.

W2.9-10107) and filed in the United States Patent and Trademark Office on May 14, 2016, is hereby incorporated by reference herein in its entirety as if fully set forth herein.

[0093] Application No. _____, entitled "QUERY DISPATCH AND EXECUTION ARCHITECTURE" (Attorney Docket No. W3.1-10061) and filed in the United States Patent and Trademark Office on May 14, 2016, is hereby incorporated by reference herein in its entirety as if fully set forth herein.

[0094] Application No. _____, entitled "COMPUTER DATA DISTRIBUTION ARCHITECTURE" (Attorney Docket No. W3.2-10087) and filed in the United States Patent and Trademark Office on May 14, 2016, is hereby incorporated by reference herein in its entirety as if fully set forth herein.

[0095] Application No. _____, entitled "DYNAMIC UPDATING OF QUERY RESULT DISPLAYS" (Attorney Docket No. W3.3-10059) and filed in the United States Patent and Trademark Office on May 14, 2016, is hereby incorporated by reference herein in its entirety as if fully set forth herein.

[0096] Application No. _____, entitled "DYNAMIC CODE LOADING" (Attorney Docket No. W3.4-10065) and filed in the United States Patent and Trademark Office on May 14, 2016, is hereby incorporated by reference herein in its entirety as if fully set forth herein.

[0097] Application No. _____, entitled "IMPORTATION, PRESENTATION, AND PERSISTENT STORAGE OF DATA" (Attorney Docket No. W3.5-10088) and filed in the United States Patent and Trademark Office on May 14, 2016, is hereby incorporated by reference herein in its entirety as if fully set forth herein.

[0098] Application No. _____, entitled "COMPUTER DATA DISTRIBUTION ARCHITECTURE" (Attorney Docket No. W3.7-10079) and filed in the United States Patent and Trademark Office on May 14, 2016, is hereby incorporated by reference herein in its entirety as if fully set forth herein.

[0099] Application No. _____, entitled "PERSISTENT QUERY DISPATCH AND EXECUTION ARCHITECTURE" (Attorney Docket No. W4.2-10089) and filed in the United States Patent and Trademark Office on May 14, 2016, is hereby incorporated by reference herein in its entirety as if fully set forth herein.

[0100] Application No. _____, entitled "SINGLE INPUT GRAPHICAL USER INTERFACE CONTROL ELEMENT AND METHOD" (Attorney Docket No. W4.3-10063)

and filed in the United States Patent and Trademark Office on May 14, 2016, is hereby incorporated by reference herein in its entirety as if fully set forth herein.

[0101] Application No. _____, entitled "GRAPHICAL USER INTERFACE DISPLAY EFFECTS FOR A COMPUTER DISPLAY SCREEN" (Attorney Docket No. W4.4-10090) and filed in the United States Patent and Trademark Office on May 14, 2016, is hereby incorporated by reference herein in its entirety as if fully set forth herein.

[0102] Application No. _____, entitled "COMPUTER ASSISTED COMPLETION OF HYPERLINK COMMAND SEGMENTS" (Attorney Docket No. W4.5-10091) and filed in the United States Patent and Trademark Office on May 14, 2016, is hereby incorporated by reference herein in its entirety as if fully set forth herein.

[0103] Application No. _____, entitled "HISTORICAL DATA REPLAY UTILIZING A COMPUTER SYSTEM" (Attorney Docket No. W5.1-10080) and filed in the United States Patent and Trademark Office on May 14, 2016, is hereby incorporated by reference herein in its entirety as if fully set forth herein.

[0104] Application No. _____, entitled "DATA STORE ACCESS PERMISSION SYSTEM WITH INTERLEAVED APPLICATION OF DEFERRED ACCESS CONTROL FILTERS" (Attorney Docket No. W6.1-10081) and filed in the United States Patent and Trademark Office on May 14, 2016, is hereby incorporated by reference herein in its entirety as if fully set forth herein.

[0105] Application No. _____, entitled "REMOTE DATA OBJECT PUBLISHING/SUBSCRIBING SYSTEM HAVING A MULTICAST KEY-VALUE PROTOCOL" (Attorney Docket No. W7.2-10064) and filed in the United States Patent and Trademark Office on May 14, 2016, is hereby incorporated by reference herein in its entirety as if fully set forth herein.

[0106] While the disclosed subject matter has been described in conjunction with a number of embodiments, it is evident that many alternatives, modifications and variations would be, or are, apparent to those of ordinary skill in the applicable arts. Accordingly, Applicants intend to embrace all such alternatives, modifications, equivalents and variations that are within the spirit and scope of the disclosed subject matter.

CLAIMS

What is claimed is:

1. A system for parsing, generating code and compiling computer data system query language code, the system comprising:

one or more hardware processors coupled to a nontransitory computer readable medium having stored thereon software instructions that, when executed by the one or more processors, cause the one or more processors to perform operations including:

obtaining, at the one or more hardware processors, computer data system query language code from an electronic computer readable data storage;

parsing, at the one or more hardware processors, the computer data system query language code to generate a computer data system query language parsed code structure;

resolving, at the one or more hardware processors, a type of one or more columns represented in the parsed code structure;

inserting, at the one or more hardware processors, resolved types into the parsed code structure;

generating, at the one or more hardware processors, computer programming language code from the computer data system query language parsed code structure;

determining, at the one or more hardware processors, whether precompiled code corresponding to the generated computer programming language code is available in a precompiled code repository stored in the electronic computer readable data storage;

when precompiled code is available in the precompiled code repository, loading, at the one or more hardware processors, the precompiled code;

when precompiled code is not available in the precompiled code repository:

compiling, at the one or more hardware processors, the computer programming language code to generate compiled computer programming language code; and

loading, at the one or more hardware processors, the compiled computer programming language code;

instantiating, at the one or more hardware processors, the loaded precompiled code or the compiled computer programming language code; and

executing, at the one or more hardware processors, the instantiated code to perform a query operation corresponding to the computer data system query language code.

2. The system of claim 1, wherein the computer data system query language code includes one or more instructions of a data system query language.

3. The system of claim 1, wherein the operations further include identifying a source of columns or query scope variables for substitution and use in computer programming language code.

4. The system of claim 1, wherein the operations further include persisting the compiled computer programming language code by storing the precompiled code in the precompiled code repository.

5. The system of claim 1, wherein determining whether precompiled code corresponding to the programming language code is available in a precompiled code repository comprises:

generating a token representing the generated computer programming language code, wherein the token includes a result of a hash function of one or more attributes of the computer programming language code;

comparing the token to one or more repository tokens in the precompiled code repository corresponding to precompiled code units; and

based on the comparing, determining whether the token matches any of the repository tokens.

6. The system of claim 1, wherein the inserting includes inserting references to data objects and variables that have been made available to the query language code.

7. The system of claim 1, wherein the operations further include repeating the resolving and inserting until any unresolved columns or variables have been resolved.

8. A method for parsing, generating code and compiling computer data system query

language code, the method comprising:

obtaining, at a hardware processor, computer data system query language code from an electronic computer readable data storage;

parsing, at the hardware processor, the computer data system query language code to generate a computer data system query language parsed code structure;

resolving, at the hardware processor, a type of one or more columns represented in the parsed code structure;

inserting, at the hardware processor, resolved types into the parsed code structure;

generating, at the hardware processor, computer programming language code from the computer data system query language parsed code structure;

determining, at the hardware processor, whether precompiled code corresponding to the generated computer programming language code is available in a precompiled code repository stored in the electronic computer readable data storage;

when precompiled code is available in the precompiled code repository, loading, at the hardware processor, the precompiled code;

when precompiled code is not available in the precompiled code repository:

compiling, at the hardware processor, the computer programming language code to generate compiled computer programming language code; and

loading, at the hardware processor, the compiled computer programming language code;

instantiating, at the hardware processor, the loaded precompiled code or the compiled computer programming language code; and

executing, at the hardware processor, the instantiated code to perform a query operation corresponding to the computer data system query language code.

9. The method of claim 8, wherein the computer data system query language code includes one or more instructions of a data system query language.

10. The method of claim 8, further comprising identifying a source of columns or query scope variables for substitution and use in computer programming language code.

11. The method of claim 8, further comprising persisting the compiled computer programming language code by storing the precompiled code in the precompiled code repository.

12. The method of claim 8, wherein determining whether precompiled code corresponding to the programming language code is available in a precompiled code repository comprises:

generating a token representing the generated computer programming language code, wherein the token includes a result of a hash function of one or more attributes of the computer programming language code;

comparing the token to one or more repository tokens in the precompiled code repository corresponding to precompiled code units; and

based on the comparing, determining whether the token matches any of the repository tokens.

13. The method of claim 8, wherein the inserting includes inserting references to data objects and variables that have been made available to the query language code.

14. The method of claim 8, further comprising repeating the resolving and inserting until any unresolved columns or variables have been resolved.

15. A nontransitory computer readable medium having stored thereon software instructions that, when executed by one or more processors, cause the one or more processors to perform operations including:

obtaining, at the one or more hardware processors, computer data system query language code from an electronic computer readable data storage;

parsing, at the one or more hardware processors, the computer data system query language code to generate a computer data system query language parsed code structure;

resolving, at the one or more hardware processors, a type of one or more columns represented in the parsed code structure;

inserting, at the one or more hardware processors, resolved types into the parsed

code structure;

generating, at the one or more hardware processors, computer programming language code from the computer data system query language parsed code structure;

determining, at the one or more hardware processors, whether precompiled code corresponding to the generated computer programming language code is available in a precompiled code repository stored in the electronic computer readable data storage;

when precompiled code is available in the precompiled code repository, loading, at the one or more hardware processors, the precompiled code;

when precompiled code is not available in the precompiled code repository:

compiling, at the one or more hardware processors, the computer programming language code to generate compiled computer programming language code; and

loading, at the one or more hardware processors, the compiled computer programming language code;

instantiating, at the one or more hardware processors, the loaded precompiled code or the compiled computer programming language code; and

executing, at the one or more hardware processors, the instantiated code to perform a query operation corresponding to the computer data system query language code.

16. The nontransitory computer readable medium of claim 15, wherein the computer data system query language code includes one or more instructions of a data system query language.

17. The nontransitory computer readable medium of claim 15, wherein the operations further include identifying a source of columns or query scope variables for substitution and use in computer programming language code.

18. The nontransitory computer readable medium of claim 15, wherein the operations further include persisting the compiled computer programming language code by storing the precompiled code in the precompiled code repository.

19. The nontransitory computer readable medium of claim 15, wherein determining whether precompiled code corresponding to the programming language code is available in a

precompiled code repository comprises:

generating a token representing the generated computer programming language code, wherein the token includes a result of a hash function of one or more attributes of the computer programming language code;

comparing the token to one or more repository tokens in the precompiled code repository corresponding to precompiled code units; and

based on the comparing, determining whether the token matches any of the repository tokens.

20. The nontransitory computer readable medium of claim 15, wherein the inserting includes inserting references to data objects and variables that have been made available to the query language code, and wherein the operations further include repeating the resolving and inserting until any unresolved columns or variables have been resolved.

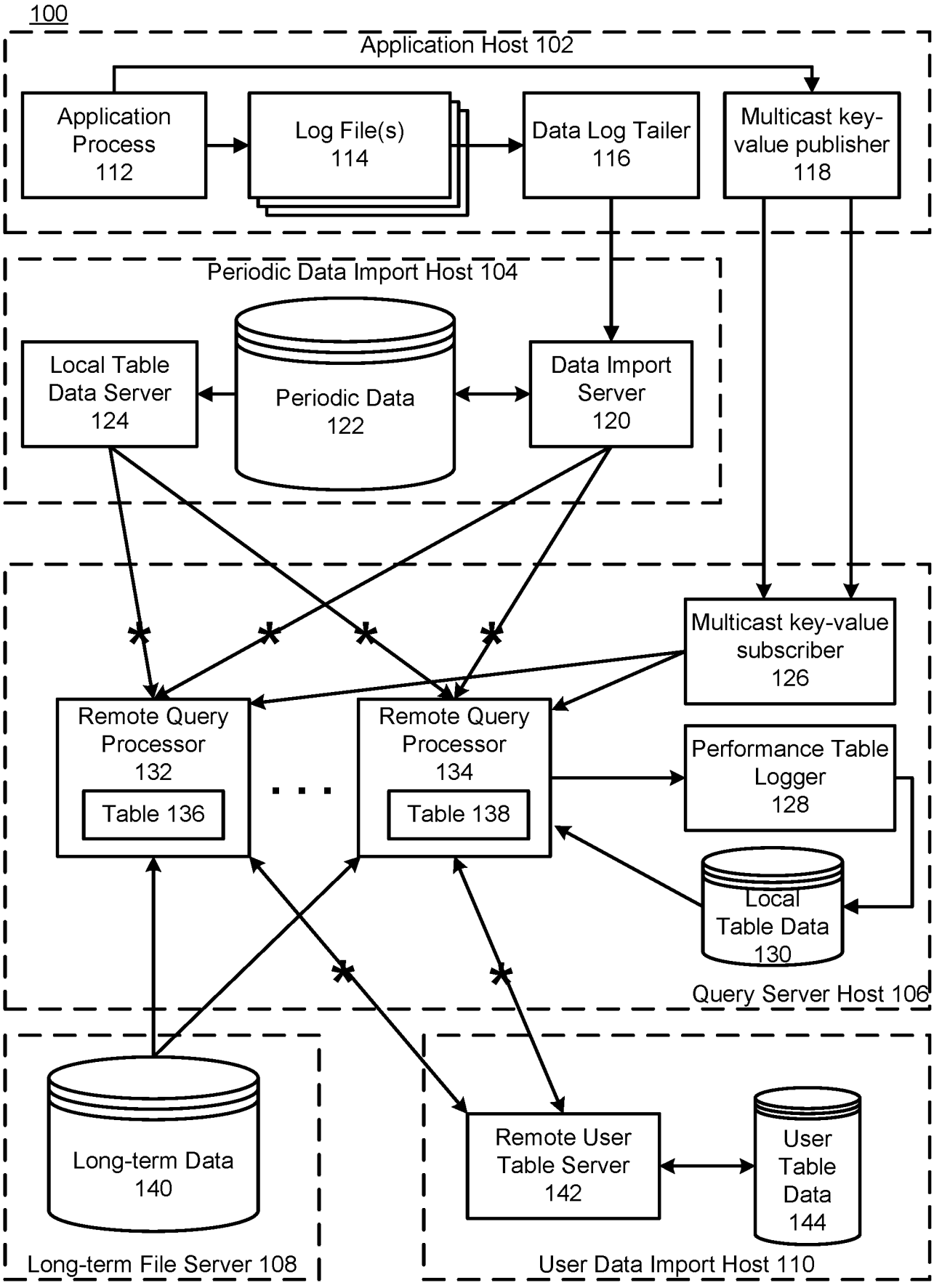
100
Application Host 102
Application Process 112
Log File(s) 114
Data Log Tailer 116
Multicast key-value publisher 118
Periodic Data Import Host 104
Local Table Data Server 124
Periodic Data 122
Data Import Server 120
Multicast key-value subscriber 126
Remote Query Processor 132
Table 136
Remote Query Processor 134
Table 138
Performance Table Logger 128
Local Table Data 130
Query Server Host 106
Long-term Data 140
Long-term File Server 108
Remote User Table Server 142
User Table Data 144
User Data Import Host 110

FIG. 1

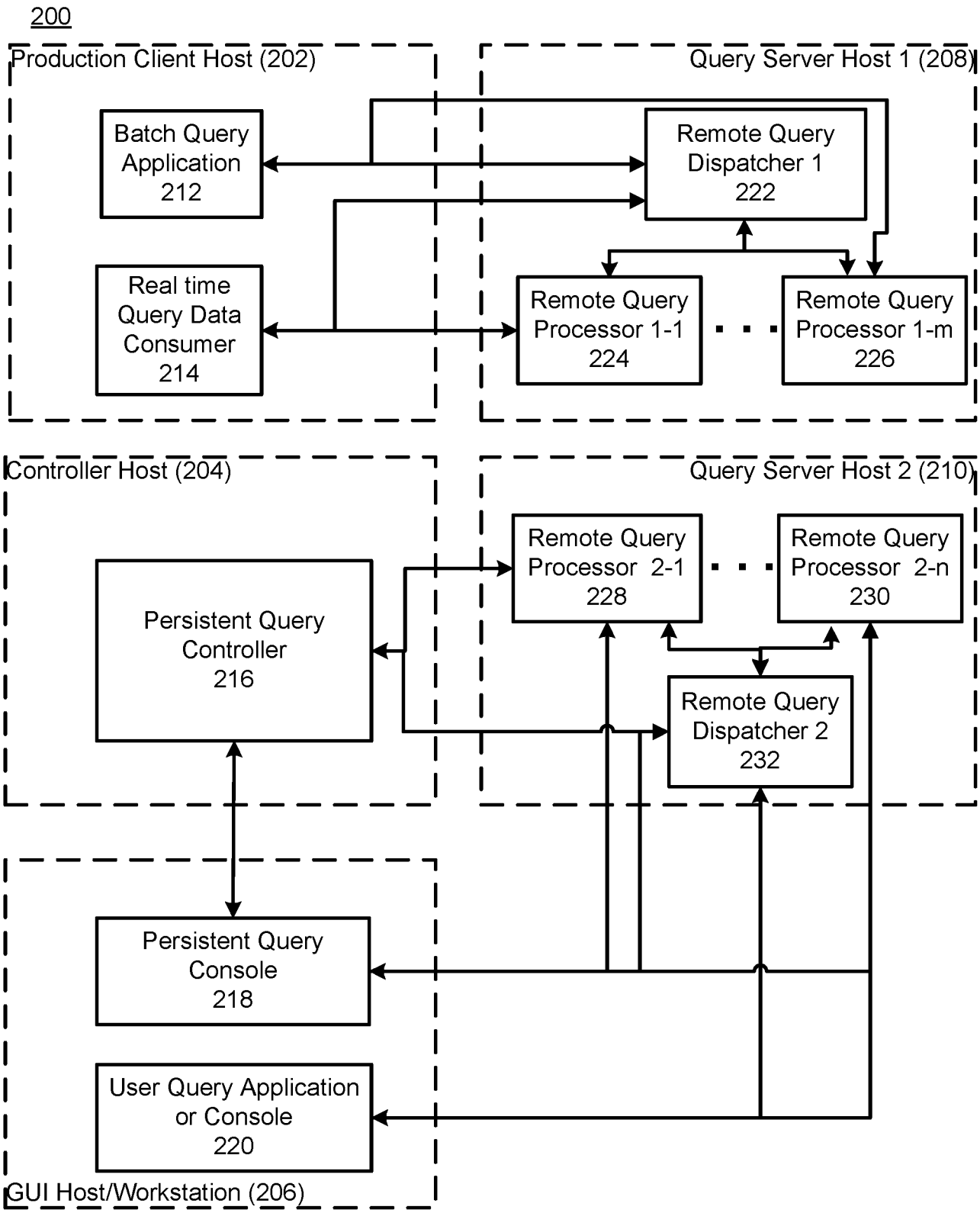
200
Production Client Host (202)
Batch Query Application 212
Real time Query Data Consumer 214
Query Server Host 1 (208)
Remote Query Dispatcher 1 222
Remote Query Processor 1-1 224
Remote Query Processor 1-m 226
Controller Host (204)
Persistent Query Controller 216
Query Server Host 2 (210)
Remote Query Processor 2-1 228
Remote Query Processor 2-n 230
Remote Query Dispatcher 2 232
Persistent Query Console 218
User Query Application or Console 220
GUI Host/Workstation (206)

FIG. 2

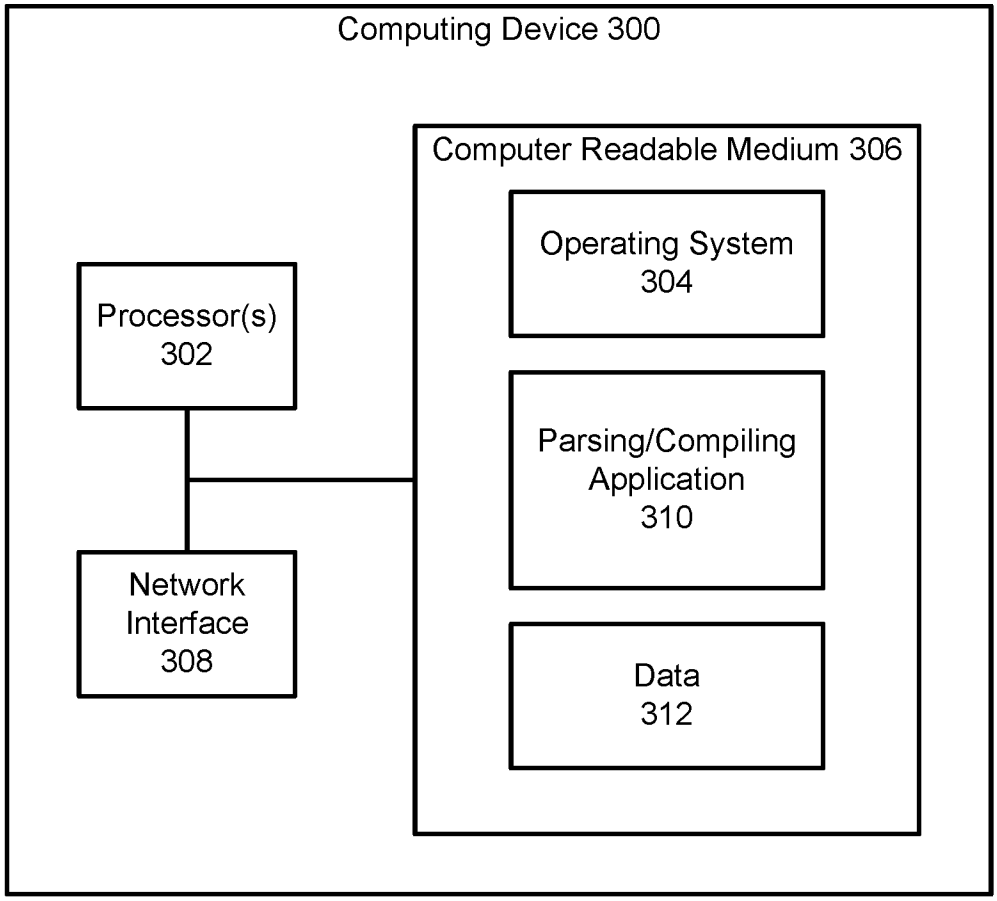
Computing Device 300
Computer Readable Medium 306
Operating System 304
Processor(s) 302
Parsing/Compiling Application 310
Network Interface 308
Data 312

FIG. 3

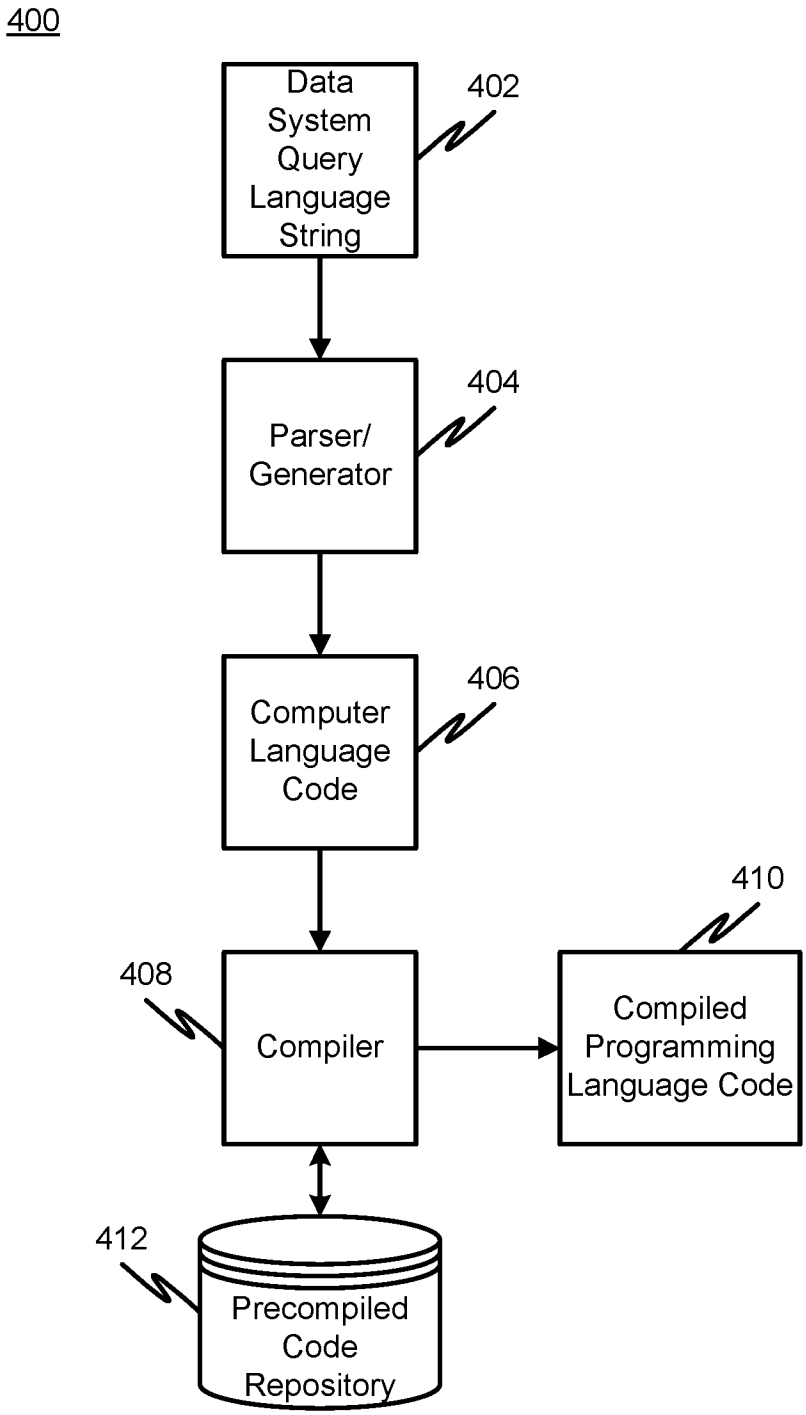
400
Data System Query Language String
402
Parser/ Generator
404
Computer Language Code
406
Compiler
408
Compiled Programming Language Code
410
Precompiled Code Repository
412

FIG. 4

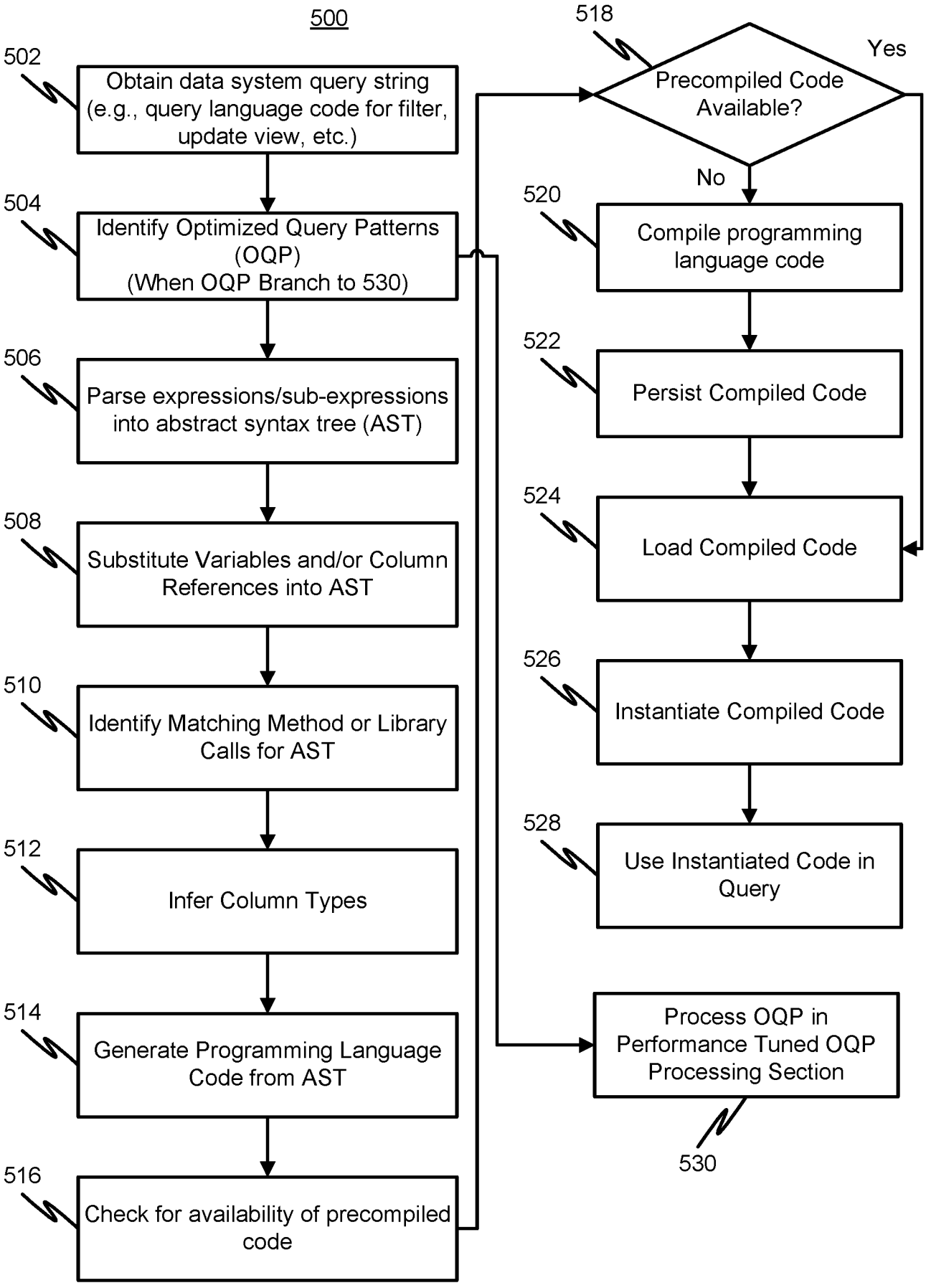
500
502
Obtain data system query string (e.g., query language code for filter, update view, etc.)
504
Identify Optimized Query Patterns (OQP) (When OQP Branch to 530)
506
Parse expressions/sub-expressions into abstract syntax tree (AST)
508
Substitute Variables and/or Column References into AST
510
Identify Matching Method or Library Calls for AST
512
Infer Column Types
514
Generate Programming Language Code from AST
516
Check for availability of precompiled code
518
Precompiled Code Available?
Yes
No
520
Compile programming language code
522
Persist Compiled Code
524
Load Compiled Code
526
Instantiate Compiled Code
528
Use Instantiated Code in Query
Process OQP in Performance Tuned OQP Processing Section
530

FIG. 5

INTERNATIONAL SEARCH REPORT

International application No.

PCT/US 2016/032589

A. CLASSIFICATION OF SUBJECT MATTER

G06F 7/00 *(2006.01)*
G06F 17/30 *(2006.01)*

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F 7/00, 17/22, 17/27, 17/30

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

PatSearch (RUPTO internal), USPTO, PAJ, K-PION, Esp@cenet, Information Retrieval System of FIPS

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 2003/0167261 A1 (INTERNATIONAL BUSINESS MACHINES CORPORATION) 04.09.2003, abstract, [0024], [0069]	1-20
Y	US 5504885 A (TEXAS INSTRUMENTS INCORPORATED) 02.04.1996, col. 5, lines 9-12, col. 9, lines 2-10	1-20
Y	US 6985904 B1 (ORACLE INTERNATIONAL CORPORATION) 10.01.2006, col. 3, lines 39-64, col. 5, lines 39-40	1-20

☐ Further documents are listed in the continuation of Box C.

☐ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
20 July 2016 (20.07.2016)	28 July 2016 (28.07.2016)

Name and mailing address of the ISA/RU:
Federal Institute of Industrial Property,
Berezhkovskaya nab., 30-1, Moscow, G-59,
GSP-3, Russia, 125993
Facsimile No: (8-495) 531-63-18, (8-499) 243-33-37

Authorized officer

A. Tokarev

Telephone No. 499-240-25-91