Office de la Propriété Intellectuelle du Canada
Un organisme d'Industrie Canada

Canadian Intellectual Property Office
An agency of Industry Canada

CA 2867279 A1 2015/04/18

(21) 2 867 279

(12) DEMANDE DE BREVET CANADIEN
CANADIAN PATENT APPLICATION

(13) A1

(22) Date de dépôt/Filing Date: 2014/10/14

(41) Mise à la disp. pub./Open to Public Insp.: 2015/04/18

(30) Priorités/Priorities: 2013/10/18 (US61/893,080);
2014/10/01 (US14/504,103)

(51) Cl.Int./Int.Cl. *G06F 17/30* (2006.01)

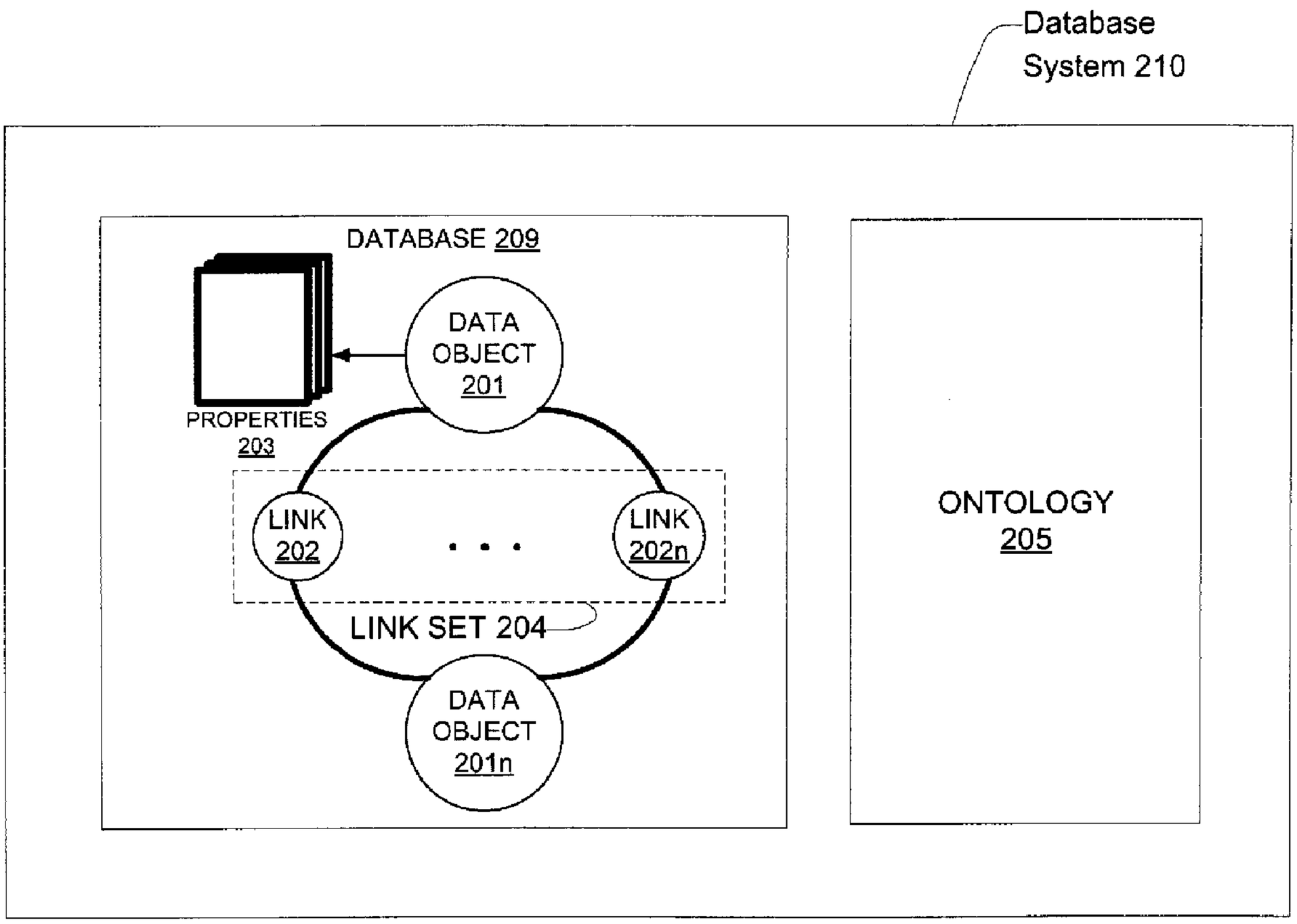
(71) Demandeur/Applicant:
PALANTIR TECHNOLOGIES, INC., US

(72) Inventeurs/Inventors:
SHANKAR, ANKIT, US;
ASH, ANDREW, US;
STOWE, GEOFF, US;
PETRACCA, THOMAS, US;
DUFFIELD, BENJAMIN, US

(74) Agent: GOWLING LAFLEUR HENDERSON LLP

(54) Titre : SYSTEMES ET INTERFACES UTILISATEURS POUR REQUETES SIMULTANEES DYNAMIQUES ET INTERACTIVES AUPRES DE MULTIPLES MAGASINS DE DONNEES

(54) Title: SYSTEMS AND USER INTERFACES FOR DYNAMIC AND INTERACTIVE SIMULTANEOUS QUERYING OF MULTIPLE DATA STORES



(57) Abrégé/Abstract:
Embodiments of the present disclosure relate to a computer system and interactive user interfaces configured to enable efficient and rapid access to multiple different data sources simultaneously, and by an unskilled user. The unskilled user may provide simple

CA 2867279 A1 2015/04/18

(21) 2 867 279

(13) A1

(57) Abrégé(suite)/Abstract(continued):
and intuitive search terms to the system, and the system may thereby automatically query multiple related data sources of different types and present results to the user. Data sources in the system may be efficiently interrelated with one another by way of a mathematical graph in which nodes represent data sources and/or portions of data sources (for example, database tables), and edges represent relationships among the data sources and/or portions of data sources. For example, edges may indicate relationships between particular rows and/or columns of various tables. The table graph enables a compact and memory efficient storage of relationships among various disparate data sources.

ABSTRACT

Embodiments of the present disclosure relate to a computer system and interactive user interfaces configured to enable efficient and rapid access to multiple different data sources simultaneously, and by an unskilled user. The unskilled user may provide simple and intuitive search terms to the system, and the system may thereby automatically query multiple related data sources of different types and present results to the user. Data sources in the system may be efficiently interrelated with one another by way of a mathematical graph in which nodes represent data sources and/or portions of data sources (for example, database tables), and edges represent relationships among the data sources and/or portions of data sources. For example, edges may indicate relationships between particular rows and/or columns of various tables. The table graph enables a compact and memory efficient storage of relationships among various disparate data sources.

SYSTEMS AND USER INTERFACES FOR DYNAMIC AND INTERACTIVE SIMULTANEOUS QUERYING OF MULTIPLE DATA STORES

TECHNICAL FIELD

Embodiments of present disclosure relate to systems and techniques for accessing one or more databases in substantially real-time to provide information in an interactive user interface. More specifically, embodiments of the present disclosure relate to user interfaces for automatically and simultaneously querying multiple different data sets and/or different electronic collections of data.

BACKGROUND

The approaches described in this section are approaches that could be pursued, but not necessarily approaches that have been previously conceived or pursued. Therefore, unless otherwise indicated, it should not be assumed that any of the approaches described in this section qualify as prior art merely by virtue of their inclusion in this section.

Traditional database queries are generated and used by skilled computer programmers to access data from a data source, such as a database. Traditional database queries are useful in many fields (for example, scientific fields, financial fields, political fields, and/or the like). Typically, a computer programmer must determine the proper format for the query based on the type of database accessed, and must determine the parameters of the query users or analysts that are familiar with the requirements of the data needed. Some man-machine interfaces for generating reports in this manner are software development tools that allow a computer programmer to write and test computer programs. Following development and testing of the computer program, the computer program must be released into a production environment for use. Thus, this approach for generating queries may be inefficient because an entire software development life cycle (for example, requirements gathering, development, testing, and release) may be required even if only one element of the query requires changing, or one aspect of the database has changed. Furthermore, this software development life cycle may be inefficient and consume significant processing and/or memory resources.

Further, traditional queries must be formatted specifically as required by the type of data source accessed. Accordingly, traditional methods of database querying have difficulties with handling queries to various types of data sources at the same time.

SUMMARY

The systems, methods, and devices described herein each have several aspects, no single one of which is solely responsible for its desirable attributes. Without limiting the scope of this disclosure, several non-limiting features will now be discussed briefly.

Embodiments of the present disclosure relate to a computer system and interactive user interfaces configured to enable efficient and rapid access to multiple different data sources simultaneously, and by an unskilled user. For example, the unskilled user may provide simple and intuitive search terms to the system, and the system may thereby automatically query multiple related data sources of different types and present results to the user. Accordingly, the system may enable efficient unskilled user interactions with complex and disparate data sources, and efficient presentation of data (including search results) to the user.

In various embodiments, data sources in the system may be efficiently interrelated with one another by way of a mathematical graph (referred to herein as a "table graph") in which nodes represent data sources and/or portions of data sources (for example, database tables), and edges represent relationships among the data sources and/or portions of data sources. For example, edges may indicate relationships between particular rows and/or columns of various tables. The table graph enables a compact and memory efficient storage of relationships among various disparate data sources. By comparison to previous methods, in which data from disparate sources of different types were usually re-copied to common data sources and/or updated to a common type, the present disclosure describes a system in which these steps need not be performed, and thus the system is more efficient from both processor usage and memory usage perspectives.

Further, embodiments of the present disclosure relate to a computer system designed to provide interactive, graphical user interfaces (also referred to herein as "user interfaces") for enabling non-technical users to quickly and dynamically generate, edit, and update search queries. The user interfaces are interactive such that a user may make selections, provide

inputs, and/or manipulate outputs. In response to various user inputs, the system automatically accesses one or more table graphs, formulates necessary database queries, traverses and queries associated data sources, obtains search results, and/or displays the search results to the user.

Various embodiments of the present disclosure enable search query generation and display in fewer steps, result in faster creation of outputs (such as search results), consume less processing and/or memory resources than previous technology, permit users to have less knowledge of programming languages and/or software development techniques, and/or allow less technical users or developers to create outputs (such as search results) than previous systems and user interfaces. Thus, in some embodiments, the user interfaces described herein are more efficient as compared to previous user interfaces, and enable the user to cause the system to automatically access and query multiple different data sources.

In some embodiments of the present disclosure, user interfaces are generated that enable a user to efficiently relate various different data sources in a mathematical graph.

In some embodiments, the system may automatically perform queries of data sources related in the table graph in parallel. Accordingly, the present system may be even more efficient over previous systems as the table graph enables parallel and automatic querying of multiple data sources so as to provide search results to the user with even greater speed and frees up processor resources for other tasks.

Further, as described herein, in some embodiments the system may be configured and/or designed to generate user interface data useable for rendering the various interactive user interfaces described. In these embodiments, the user interface data may be used by the system, and/or another computer system, device, and/or software program (for example, a browser program), to render the interactive user interfaces. The interactive user interfaces may be displayed on, for example, electronic displays (including, for example, touch-enabled displays).

According to an embodiment, a search system is disclosed that is configured to execute a search query, the search system comprising: one or more computing devices having hardware processors configured to execute instructions in order to: access a first data store of a first type storing at least a first data table; access a second data store of a second

type different than the first type, the second data store storing at least a second data table; execute a table graph having a first node associated with the first data table, a second node associated with the second data table, and a link between the first node and the second node, the link indicating a first field of the first data table that is associated with a second field of the second data table, said executing comprising: looking up first information associated with a search query in the first data store; look up second information associated with the first information in the second data store; provide the second information for display or processing by the search system.

According to an aspect, at least one of the first field and the second field are full-text searchable.

According to another aspect, the link between the first node and the second node is a bi-directional link.

According to yet another aspect, the first type and the second type each include one or more of a relational data store, an object-oriented data store, a proprietary data store, a file-based data store, a hierarchical data store, a network data store, and an elastic-search data store.

According to another aspect, the table graph is executed via an application programming interface configured to format the second information as one or more data objects, wherein the data objects further comprise attributes and values defined by a user.

According to yet another aspect, the search query includes one or more regular expression rules.

According to another aspect, the table graph further comprises a third node associated with a third data table and wherein executing the table graph further comprises concurrently looking up the first information in the first data table and looking up a third information in the third data table.

According to yet another aspect, the lookups in the first data table and the third data tables are executed concurrently in response to the one or more computing devices automatically determining that the lookups are suitable for concurrent execution based on at least one of: query dependency, query complexity, history of query execution duration, usage

statistics of the first, second, and/or third data stores, and/or predicted usage demand of the first, second, and third data stores.

According to another embodiment, a table graph system is disclosed that is configured to electronically communicate with at least one data store, the table graph system comprising: one or more physical computing devices having hardware processors configured to execution instructions in order to: receive instructions to add a first data table as a first node in a table graph; receive instructions to add a second data table as a second node in the table graph, wherein the first data table and the second data table are of different types; upon receiving instructions to add a relationship between the first and second nodes, add the relationship, wherein the relationship indicates a first field of the first data table is to be associated with a second field of the second data table, wherein the table graph is executable in order to access data stored in the first data table and subsequently data in the second data table based at least partly on the data stored in the first data table.

According to an aspect, the one or more physical computing devices are further configured to: receive instructions to add a third node linked with the first node and/or the second node; receive instructions to indicate the first node and the third node as concurrently processable, wherein the first node and the third node are executable concurrently in response to execution of the table graph.

According to another aspect, the one or more physical computing devices are further configured to: receive instructions to add a third node linked with the first node and/or the second node; receive instructions to execute at least two of the nodes concurrently; automatically determine an order of executing the at least two of the nodes concurrently.

According to yet another aspect, the one or more physical computing devices are further configured to automatically add one or more relationships between the first and second nodes.

According to another aspect, the one or more physical computing devices are further configured to receive instructions to change the relationship between the first and second nodes from a one-directional link to a bi-directional link.

According to yet another aspect, the first data table and the second data table are located remotely from each other.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 illustrates one embodiment of a database system using an ontology.

Figure 2 illustrates one embodiment of a system for creating and accessing a table graph.

Figure 3 illustrates one example of executing an example table graph.

Figure 4A illustrates two stages of executing the example table graph of Figure 3 with example data tables.

Figure 4B illustrates an example interface for entering search terms for a search that is executed using one or more table graphs.

Figure 5A is a flowchart depicting an illustrative process of executing a search using a table graph.

Figures 5B is a flowchart depicting an illustrative process of generating of a table graph.

Figure 6A illustrates an example interface that is configured to enable users to create a table graph and configure properties of the table graph.

Figure 6B illustrates an example interface that is configured to allow users to drag and drop tables from various data sources into the interface in order to create a table graph.

Figure 7 illustrates an example interface that displays a table graph that includes nodes that can be executed in parallel.

Figure 8 is a flowchart depicting an illustrative process of executing select nodes of a table graph in parallel.

Figure 9 is a block diagram illustrating one embodiment of a computer system with which certain methods and modules discussed herein may be implemented.

DETAILED DESCRIPTION OF CERTAIN EMBODIMENTS

Terms

In order to facilitate an understanding of the systems and methods discussed herein, a number of terms are defined below. The terms defined below, as well as other terms used herein, should be construed to include the provided descriptions, the ordinary and customary meaning of the terms, and/or any other implied meaning for the respective terms. Thus, the

descriptions below do not limit the meaning of these terms, but only provide exemplary definitions.

Ontology: Stored information that provides a data model for storage of data in one or more databases. For example, the stored data may comprise definitions for object types and property types for data in a database, and how objects and properties may be related.

Database: A broad term for any data structure for storing and/or organizing data, including, but not limited to, relational databases (Oracle database, mySQL database, etc.), spreadsheets, XML files, and text file, among others. It is also called a data store or a data structure herein.

Data Object or Object: A data container for information representing specific things in the world that have a number of definable properties. For example, a data object can represent an entity such as a person, a place, an organization, a market instrument, or other noun. A data object can represent an event that happens at a point in time or for a duration. A data object can represent a document or other unstructured data source such as an e-mail message, a news report, or a written paper or article. Each data object may be associated with a unique identifier that uniquely identifies the data object. The object's attributes (e.g. metadata about the object) may be represented in one or more properties.

Object Type: Type of a data object (e.g., person, event, or document). Object types may be defined by an ontology and may be modified or updated to include additional object types. An object definition (e.g., in an ontology) may include how the object is related to other objects, such as being a sub-object type of another object type (e.g. an agent may be a sub-object type of a person object type), and the properties the object type may have.

Properties: Attributes of a data object that represent individual data items. At a minimum, each property of a data object has a property type and a value or values.

Property Type: The type of data a property is, such as a string, an integer, or a double. Property types may include complex property types, such as a series data values associated with timed ticks (e.g. a time series), etc.

Property Value: The value associated with a property, which is of the type indicated in the property type associated with the property. A property may have multiple values.

Link: A connection between two data objects, based on, for example, a relationship, an event, and/or matching properties. Links may be directional, such as one representing a payment from person A to B, or bidirectional.

Link Set: Set of multiple links that are shared between two or more data objects.

Table Graph: Set of multiple nodes and edges between two or more of the nodes. A node in a table graph may represent a data table or a portion of a data table. An edge in a table graph may represent a relationship between two nodes in the table graph. A relationship may represent, for example, a mapping between a data field of a node and a data field of another node.

Object Centric Data Model

To provide a framework for the following discussion of specific systems and methods described herein, an example database system 210 using an ontology 205 will now be described. This description is provided for the purpose of providing an example and is not intended to limit the techniques to the example data model, the example database system, or the example database system's use of an ontology to represent information.

In one embodiment, a body of data is conceptually structured according to an object-centric data model represented by ontology 205. The conceptual data model is independent of any particular database used for durably storing one or more database(s) 209 based on the ontology 205. For example, each object of the conceptual data model may correspond to one or more rows in a relational database or an entry in Lightweight Directory Access Protocol (LDAP) database, or any combination of one or more databases.

FIG. 1 illustrates an object-centric conceptual data model according to an embodiment. An ontology 205, as noted above, may include stored information providing a data model for storage of data in the database 209. The ontology 205 may be defined by one or more object types, which may each be associated with one or more property types. At the highest level of abstraction, data object 201 is a container for information representing things in the world. For example, data object 201 can represent an entity such as a person, a place, an organization, a market instrument, or other noun. Data object 201 can represent an event that happens at a point in time or for a duration. Data object 201 can represent a document or other unstructured data source such as an e-mail message, a news report, or a written paper or

article. Each data object 201 is associated with a unique identifier that uniquely identifies the data object within the database system.

Different types of data objects may have different property types. For example, a "person" data object might have an "Eye Color" property type and an "Event" data object might have a "Date" property type. Each property 203 as represented by data in the database system 210 may have a property type defined by the ontology 205 used by the database 209.

Objects may be instantiated in the database 209 in accordance with the corresponding object definition for the particular object in the ontology 205. For example, a specific monetary payment (e.g., an object of type "event") of US$30.00 (e.g., a property of type "currency") taking place on 3/27/2009 (e.g., a property of type "date") may be stored in the database 209 as an event object with associated currency and date properties as defined within the ontology 205.

The data objects defined in the ontology 205 may support property multiplicity. In particular, a data object 201 may be allowed to have more than one property 203 of the same property type. For example, a "person" data object might have multiple "Address" properties or multiple "Name" properties.

Each link 202 represents a connection between two data objects 201. In one embodiment, the connection is either through a relationship, an event, or through matching properties. A relationship connection may be asymmetrical or symmetrical. For example, "person" data object A may be connected to "person" data object B by a "Child Of" relationship (where "person" data object B has an asymmetric "Parent Of" relationship to "person" data object A), a "Kin Of" symmetric relationship to "person" data object C, and an asymmetric "Member Of" relationship to "Organization" data object X. The type of relationship between two data objects may vary depending on the types of the data objects. For example, "person" data object A may have an "Appears In" relationship with "Document" data object Y or have a "Participate In" relationship with "Event" data object E.

As an example of an event connection, two "person" data objects may be connected by an "Airline Flight" data object representing a particular airline flight if they traveled together on that flight, or by a "Meeting" data object representing a particular meeting if they both attended that meeting. In one embodiment, when two data objects are connected by an event,

they are also connected by relationships, in which each data object has a specific relationship to the event, such as, for example, an "Appears In" relationship.

As an example of a matching properties connection, two "person" data objects representing a brother and a sister, may both have an "Address" property that indicates where they live. If the brother and the sister live in the same home, then their "Address" properties likely contain similar, if not identical property values. In one embodiment, a link between two data objects may be established based on similar or matching properties (e.g., property types and/or property values) of the data objects. These are just some examples of the types of connections that may be represented by a link and other types of connections may be represented; embodiments are not limited to any particular types of connections between data objects. For example, a document might contain references to two different objects. For example, a document may contain a reference to a payment (one object), and a person (a second object). A link between these two objects may represent a connection between these two entities through their co-occurrence within the same document.

Each data object 201 can have multiple links with another data object 201 to form a link set 204. For example, two "person" data objects representing a husband and a wife could be linked through a "Spouse Of" relationship, a matching "Address" property, and one or more matching "Event" properties (e.g., a wedding). Each link 202 as represented by data in a database may have a link type defined by the database ontology used by the database.

Figure 2 is a block diagram illustrating one embodiment of a system for creating and accessing a table graph. A table graph, as used herein, defines relationships between multiple data tables from one or more data sources. The data table, or a portion of a data table, may be represented as nodes and the relationships between nodes as edges between/among two or more of the nodes. The edges (e.g., relationships) between two nodes may represent, for example, a mapping between particular data fields of nodes.

As illustrated in Figure 2, a variety of database types may be data sources of a table graph. In the embodiment of Figure 2, a table graph API 248 is configured to interface between the table graph 250 and multiple data sources, such as the data sources illustrated in Figure 2, including a proprietary database 240, an elastic search database 242, a SQL database 244, an ODBC database 246, an object-oriented database (not shown in this figure),

a file-based data store (not shown in this figure), a key-value store (not shown in this figure) and so forth. The table graph API 248 is capable of parsing data from various heterogeneous databases using one or more parsers and/or transformers. In some embodiments, a parser may communicate with a data source, such as an external data store, an internal data store, a proprietary database, etc., and parse the data received from the data source. In some embodiments, a transformer will further transform the data into a format that can be easily read by the table graph API 248, such as an object or another type of data. Depending on the embodiment, if there is an unspecified data source that the table graph API 248 has not previously interacted with, the table graph API 248 may first automatically detect or determine a set of rules to parse data from the unspecified source, and/or a user may manually provide translation data, so that data sources of that type may later be accessed by table graphs.

The data access system 252 may be used to generate table graphs, such as table graph 250. Users such as data analysts, database administrators, IT administrators, testing engineers, QA engineers, database and/or software developers, who have sufficient privilege to interact with the data access system 252 may generate table graphs. They may also share the generated table graphs with other end users, analysts, engineers, administrators, and so forth. An end user, a data analyst, or other users who do not have to be familiar with the underlying system, may be able to easily interact with table graphs through the data access system 252.

In the example in Figure 2, a user 254 may, through the data access system 252, submit a request to add a first data table as a first node in a table graph 250. The user 254 may further add other data tables as nodes in the table graph 250. The user 254 may also specify set of relationships to add between the tables as edges of the table graph. In the example of table graph 250, five nodes A, B, C, D, and E, which may each be associated with different tables possibly from multiple data sources (or some may represent a common table) are added to table graph 250. The edge between nodes A and B is a bi-directional edge, where the direction(s) of edges indicate the direction of data flow between nodes. The other edges in the table graph 250 are one directional edges. A bi-directional edge may represent that there is a two-way relationship between a data field in one node and a data field in another node.

For example, the ID field of a first node may be mapped to the fromID field in a second node. In addition, the toID field of the second node may be mapped to the ID field of the first node. However, in some embodiments, a bi-directional edge may also be represented as two one-directional edges. A one-directional edge represents a one-way relationship from a data field in one node to a data field in another node, which, in some situations, may mean that data from one data field in one node may be used to match a data field in another node. In some embodiments, the user 254 that creates the table graph 250 may be a system administrator, database administrator, or a database or system developer. In some other embodiments, the user 254 may be an end-user or a user of the table graph system. In some situations, there may be a user that creates the table graph 250 and other users that have access to the table graph 250 or have privileges to edit, remove, and/or execute queries using the table graph 250. In some embodiments, applications may also interact with data access system 252. For example, a data quality monitoring system may need to query a table graph and process the returned results. The application 256 in this example would be the data quality monitoring system. In another example, the application 256 may be a part of an integrated system that detects fraud, security breaches, and so forth, and the application 256 may receive search queries and/or instructions to create and/or configure various table graphs for various purposes.

Figure 3 illustrates one example of a process of executing a table graph based on a user-specified request. In this example, two data tables may come from a data store 301 or the data store 301 and another data store in an external data store. The two data tables are example Customer Info 307 and Transactions 309. The two data tables have been added as nodes in the table graph shown in Figure 3. The customer info node 303 represents the Customer Info data table 307 and the transactions node 305 represents the Transactions data table 309.

As shown in the example in Fig. 3, the Customer Info table 307 (also referred to in Figure 3 as "T1") includes three data fields: ID, Customer, and Occupation. The ID field is represented in this example as numbers such as 213, 214, 215, etc. The Customer field may store names of the customers, such as "John Doe," "Jane Doe," and "James Bond," and so forth. The Transactions table (also referred to as "T2" in Figure 3), includes five data fields:

Trans. No., fromID, toID, Item, and Cost. The fromID and toID fields in the table represent the identities of the people who are involved in the transaction in a given row in the Transactions table. For example, Trans. No. 90 is a transaction from a person 213 to a person 215 that involves a printer. The printer's cost is $107.99.

Two edges, which can also be represented in some embodiments as a single bi-directional edge, connect the customer info node 303 and the transactions node 305. From the customer info node 303, an edge represents a relationship between the ID field in the customer info node 303 and the fromID field in the transactions node 305. From the transactions node 305, an edge represents a relationship between the toID field in the transactions node 305 and the ID field in the customer info node 303. With the edges establishing table lookups, a user may be able to quickly search for transactions and identities of the people involved in the transactions without performing the intermediate search steps.

Figure 4A illustrates two stages of executing the example table graph of Figure 3 with example data tables. The same underlying data tables as used in Figure 3, Customer Info 307 and Transactions 309, are used in this example. For the purposes of clear illustration, the nodes in the table graph are labeled with different reference numbers. The customer info node 401 and transactions node 402 are connected as in Figure 3, by two edges.

In one example, a user may wish to search for all the transactions from John Doe to James Bond. Traditionally, such a query may have to be carried out in several steps, possibly using different search functionality for different data sources, and/or using a complicated search term.

In the embodiment of Figure 4A, using a table graph the search may be significantly easier for the user to execute because intermediate steps of parsing query results and making further queries based on previous queries are transparent and hidden from the user. After creating the table graph with the two tables as nodes and edges established between the two tables, queries may be easily carried out. The received search query may be received by the data access system 252 (Figure 2) and carried out by accessing multiple data sources via the table graph API 248, for example. Because the ID field in the customer info node 401 is linked to the fromID field in the transactions node 401 and the toID field in the transactions node 401 is linked to the ID field of the customer info node 307, the table graph search may

be executed directly, which may involve sending over the ID of 213, and receiving a set of transactions satisfying the search criteria (James Bond, which corresponds to toID 215, and the transaction detail: Trans. No. 00090, from ID 213, toID 215, Item printer, Cost $107.99). In some embodiments, the results will be parsed and presented to the user as an object or a set of objects. The user, however, does not need to parse any intermediate results such as searching for the ID related to a name, searching for a name that is related to an ID, or searching for a transaction that is related to the IDs.

In the second stage of the search, the table graph may automatically repeat the table graph using the customer information returned from a previous execution of the table graph, such as to not only find transactions initiated by James Bond, but to also find transactions initiated by others involved in transactions with James Bond. Traditionally, such a query may have to be carried out in several steps or using a complicated search term. Through the use of the table graph, rather than the user needing to parse the results and find out that the toID of the transaction involving James Bond (fromID 215) is 214, and then executing queries of the various tables to find transactions involving ID 214, the recursive searching according to the table graph arrangement is automated. For example, the customer ID of 215 returned in the first stage (top half of Figure 4A) of the search, which corresponds to James Bond, may be used in a second execution of the table graph (bottom half of Figure 4A) in order to receive a set of transactions satisfying the search criteria (Jane Doe, which corresponds to ID 214, and the transaction detail: Trans. No. 00091, from ID 215, toID 214, Item rice, Cost $4.50). In some embodiments, the results will be parsed and presented to the user as an object or a set of objects. Such recursive execution of table lookups (and other features) are easily implemented using table graphs.

Figure 4B illustrates an example interface for entering search term for a search that is executed using a table graph. The example user interface 850 may be generated and presented to a user, such as a data analyst that wishes to find data on a certain data object, including multiple levels of interactions with the data object that may be indicated in various data tables of various formats. In the example user interface 850, the field to be searched is Customer's Name, as shown in text field 855. The specific query includes the following search term: John D*, which the user may enter in the search box 860. The wildcard "*"

represents that any character may be deemed a match. For example, a customer's name that is "John Doe" and a customer's name that is "John Davis" both satisfy this search criteria.

Depending on the embodiment, results of a search may be displayed in the example user interface 850 or in a separate interface. For purposes of illustration, Figure 8 illustrates search results that may be returned by executing the table graph of Figures 3-4 (although additional results may be obtained if recursive execution of the table graph is continued). In the example as shown, a direct hit is displayed: "A transaction for $107.99 was found between John Doe and James Bond for a Printer (Transaction No. 00090)." In addition, an indirect hit (e.g., a transaction of an indication associated with John D*, rather than John D* himself) may also be displayed, such as the one in the example user interface 850: "A transaction for $4.50 was found between James Bond and Jane Doe for Rice (Transaction No. 00091)." In this example, the indirect hit may be found by the search because John Doe has transacted with James Bond, who also transacted with Jane Doe.

In some embodiments, instead of displaying search results in formatted text, the user interface 850 may display all or a subset of the search results as data field values, unparsed results, objects, and/or files. The results may also be offered as downloadable files formatted in various ways for the convenience of the user.

Example Table Graph Methods

Figure 5A is a flowchart depicting an illustrative process of executing a search using a table graph. The process of Figure 5A may be performed by the data access system 252 and/or the API 248 in response to input from a data analyst , for example, such as part of a data analysis software platform. However, the process may be performed by other computing systems. Depending on the embodiment, the method of Figure 5A may include fewer or additional blocks and the blocks may be performed in an order that is different than illustrated.

The process 500 begins at block 503, wherein the data access system 252 accesses a table graph, such as a table graph selected by the user or a default table graph associated with a particular search functionality that the user wishes to perform (whether or not the user even knows that table graphs are being used to execute the search functionality). For purposes of the example in Figure 5A, a table graph having two nodes (such as in Figures 3-4) is

assumed, although table graphs may include any quantity of nodes in various configurations. In some embodiments, accessing the table graph may also include obtaining information regarding the nodes and edges in the table graph, such as the names, properties, relations, index types, and data fields involved in the nodes and edges.

The process 500 then proceeds to block 505 and a first data store storing data of a first type is accessed. Depending on the embodiment, the first data store may be a database of several different types, such as a proprietary database, a SQL database, an elastic search database, an ODBC database, a JDBC database, an open source database, and so forth. Moreover, the data of the first type as stored in the first data store may be of any type. For example, the data may be of one or more following types: string, float, long, integer, binary, Boolean, text, object, BLOB, BigInt, char, numeric, Date, DateTime, real, SmallInt, Time, Timestamp, Varbinary, Varchar, and so forth. The structure of the data does not need to be specified before the data is accessed. For instance, in some embodiments, if an unknown data structure is encountered, the API 248 may be programmed to analyze and find out the data structure. In some embodiments, the API 248 may notify an administrator or user that an unknown data structure is encountered and receive the administrator or user's instructions.

The process 500 then proceeds to block 510 and a second data store storing data of a second type is accessed. The second data store may be a data store that is either internal or external to the first data store. For example, the second data store may be a database maintained by the same or a different organization at a different location as the first data store. Moreover, the second data store may store data that is of a different type than the first store. For example, if the first data store is a SQL database, the second data store may be a database of a proprietary type, and the two data stores may or may not be able to directly share query terms with each other. For example, the first data store may be a data store maintained by Company A that stores information related to consumers, including their names, identities, and so forth. The first data store may be implemented as a MySQL database. The second data store may be a data stored maintained by a department store B. The second data store may use the Microsoft Access format to store its transaction information related to department store customer transactions. In some embodiments, the first

and second data stores are not accessed (blocks 505, 510) until data is actually requested from the data stores, such as when the table graph is executed (block 520).

The process 500 then proceeds to block 515 and a search query is received. In some embodiments, the search query may be submitted by a user through an application. In some other embodiments, the user may directly interact with a table graph system. The search query may be processed by the data access system 252, which interacts with the table graphs and may be further be in contact with the table graph API 248 in order to access the various data sources.

The process 500 then proceeds to block 520 and the search query may be executed using the table graph. In some embodiments, the data access system 252 and/or the API 248 may analyze the search query to parse information such as the nodes and edges relevant to this search query. The data access system 252 and/or the API 248 may also generate one or more queries to the relevant data stores, which may include information such as search terms, data fields, name of the data tables, etc. Because the one or more queries formatted by the data access system 252 are transparent to the user, the user does not need to specify the various search queries (e.g., the exact query language used by the various data stores), details regarding how to join tables, how to parse intermediate results, or other details regarding the actual search execution. Moreover, as discussed previously, if the search query involves the look up of more than one tables (nodes) and the operation to be performed on one table depends on the intermediate results received from another table, such details may be transparent to the user. The user does not need to parse the intermediate results obtained at each node of the search query.

Figures 5B is a flowchart depicting an illustrative process of generating a table graph. The process of Figure 5B may be performed by the data access system 252 by a system administrator or data analyst, for example, such as part of a data analysis software platform. However, the process may be performed by other computing systems. Depending on the embodiment, the method of Figure 5B may include fewer or additional blocks and the blocks may be performed in an order that is different than illustrated.

For purposes of the example in Figure 5B, a table graph having two nodes (such as in Figures 3-4) is generated, although table graphs may include any quantity of nodes in various

configurations. The process 550 begins at block 555, wherein instructions to add a first table as a node are received from a user or an administrator. The first table may be physically located in a database that is remote from the user. Although the word "table" is used, the data in the first table may be in any other data container and/or formatter, either structured or unstructured. For example, data in the table may be in an XML-like format or a user-defined format. The data may also be in a traditional database format, an object-oriented database format, and so forth. In some embodiments, the instructions to add a first table to a table graph as a node may also include additional information such as an alias for the node. For example, an alias for the customer info node 303 (Figure 3) may be "Domain1.DB1.T1." The received instructions may also include, in some embodiments, information regarding how the data in the first table may be structured.

The process 550 then proceeds to block 560 and instructions to add another table as a node is received. In some embodiments, the additional table that is added to the table graph may be in the same database as the first table. However, in some other instances, the additional table may belong to a database that is not directly related to the first table. It could even be of a type that cannot normally directly communicate with the first table because they might be of different database types.

The process 550 then proceeds to block 565 and instructions to add a relationship between the tables are received. Depending on the embodiment, the received relationship may be of a type that is bi-directional or one-directional. When the received instructions are to create a relationship that is bi-directional, a bi-directional edge between the tables are added to the table graph. In some other embodiments, instead of creating a bi-directional edge, two one-directional edges are created and added to the table graph. The edges between the tables denote the relationship that is added to the table graph. One common type of relationship that may be added between the two nodes in a table graph is a mapping. For example, the ID column in one node may be mapped to the toID column in another node. In another example, the name column in one node may be mapped to the full_name column in another node.

The creation of an edge may not indicate that there is a one-to-one relationship between the data records stored in the two nodes. Instead, a variety of scenarios may exist.

For example, the ID column in one node may correspond to the toID column in another node, but there may be multiple records with the same toID values in the other node. Moreover, the ID column in the one node may be sequential, unique, and non-negative integer values. The toID column in the other node may have redundancy and/or non-integer values. Details such as these may be transparent to a user who queries the table graph, which adds to the efficiency and ease of use to the system.

As another example, the name column (or any other field) in one node may contain both first name and last name (or other combinations of data). The full_name column in the other node, however, may have incomplete data records, such as "James B." instead of "James Bond." The edge between these two nodes, therefore, may, based on user request, include an exact mapping of data records in the two nodes, or a mapping that include partial matches or accommodate data ranges, elastic matches, and so forth.

The process 550 then proceeds to a decision block 570 which determines whether additional tables may be added as nodes to the table graph. If the user wishes to add another table as a node to the table graph, then the answer to the question in the decision block 570 is yes. In some embodiments, instructions may be received from an administrator or a user with sufficient privilege to add a plurality of tables as nodes in a table graph, then the answer to the question in the decision block 570 is also yes. Accordingly, the process 550 proceeds to block 560, wherein instructions to add another table as a node in the table graph is received. The process 550 then proceeds to block 565 wherein the computing system receives instructions to add a relationship between nodes in the table graph.

However, if the user does not want to add more nodes to the table graph or if all the tables have been added as nodes as instructed, then the answer to the question in the decision block 570 is no. Accordingly, the process 550 proceeds to block 575, wherein the table graph is built and stored. In some embodiments, the table graph may be built and stored in a local computing device of the user or administrator that creates the table graph, such as in a computer, a tablet device, a handheld device, and so forth. In some other embodiments, the table graph may be built and stored in a remote computing device.

Moreover, building the table graph may include indexing fields (and/or columns) that may be relevant to edges in the table graph and/or fields that should be indexed according to

the instruction of a user. For example, in the customer_info node 303 (Figure 3), an index may be created for the ID column. Depending on the embodiment, various types of indexing methods and indices may be used. In addition, the Customer column may also be indexed.

Depending on the embodiments, the indexing methods used may present full text searchability and matching capability for regular expression and fuzzy matches. For example, instead of requiring an exact match to a customer's name, "James Bond" may be considered a match for a search term "James B". Also, the user may specify that a search should satisfy the requirement of "Jame*B*d", wherein the wildcard represents any character. Additional regular expression type of matching capabilities may also be accommodated and made possible by the indexing methods and/or query techniques used by a table graph.

Moreover, in some embodiments, the edges between the nodes in a table graph are tolerant of data quality issues. For example, instead of "James Bond," a data record in the database may include the name "Jomes Bond" (with a typo in "Jomes"). A fuzzy matching capability may identify this record as potentially the same as "James Bond," and the edge between two tables in which a name field is used as part of a relationship may still identify a data record containing the record "Jomes Bond."

In another example, the table graph may automatically perform record matches in various ways. For example, if a phone number is given as 123-456-7890, this may automatically be matched with other formats for phone numbers such as 123.456.7890, (123) 456-7890, and/or the like.

In one embodiment, nodes and/or edges may be added via command line instructions. For example, a node in a table graph may be added using syntax such as: Node customer = new CustomerNode(). In another embodiment, a node be also added using syntax such as: graph.addNode (customer). These syntaxes are presented for illustrative purposes only.

Example User Interfaces

Figure 6A illustrates an example interface that is configured to enable users to create a table graph and configure properties of the table graph. A user interface such as the example user interface 600 may be generated and presented to a user and allow the user to add various data sources by pressing add data sources button 615. As shown in the example interface 600, one data source, db1, has already been added. The user interface 600 may also

allow a user to add various tables as nodes by selecting them from a drop-down menu such the drop-down menu 620 and/or some other interface that shows tables available for selection, such as from the one or more selected data sources. Alternatively, the user interface may also allow a user to create a new node directly with the user interface by pressing the create new node button 625. Depending on the embodiment, once the user creates a new node, a new table is created in the underlying data store. By creating a new feature using the new node button 625, the user may conveniently add data to a table graph. For example, while analyzing data regarding James Bond, the user may think of other data that might also be relevant to the analysis, which is not currently in the table graph or in a known data store. Therefore, the user may add the table using the new node button 625. This feature allows users to interact with data in a table graph directly and dynamically, which may help yield more useful analysis results.

The user interface 600 may also allow users to configure the edge types between/among the nodes in the table graph. For example, the example user interface of Figure 6 shows nodes 650 and 655 selected by the user (indicated by the bold outline of the nodes in this example) such that a relationship between those selected nodes can be created and/or updated. In this example, a user may choose to add a bi-directional edge between node 650 (node "A") and node 655 (node "B") by dragging an edge between two nodes in the table graph. The type of an edge can also be configured and/or edited. For example, a user may choose to change a bi-directional edge into a one-directional edge. A user may also remove an existing edge.

The user interface 600 may also allow a user to select certain fields to index in order to complete the configuration of an edge. For example, the edge between nodes A 650 and B 655 may be configured to be between the "Last Name" field in node A 650 and the "Surname" field in node B 655. In the example of Figure 6, with nodes 650 and 655 selected, drop down menus 640 and 645 are populated with fields of the respective tables (associated with nodes 650 and 655) such that the user can selected corresponding index nodes of the two tables. For example, a user may choose to change the relationship between the two nodes by choosing the appropriate data field in a drop down menu 640 (for node A 650) and the drop

down menu 645 (for node B 645). In some other embodiments, other types of interface elements may be generated and presented in order for users to choose the data fields to index.

Figure 6B illustrates another example interface that is configured to allow users to drag and drop tables from various data sources into the interface to create a table graph. The example user interface 750 may be generated and presented to a user, such as an administrator and/or analyst that wishes to develop multi-source search logic. The example user interface 750, as shown, illustrates representations of two separate data stores: DB1 760, which is a proprietary database, and DB3 770, which is a SQL database. As shown, three tables from DB1 760 are available for selection as nodes in a table graph: T1 761 (User table), T2 762 (Background table), and T3 763 (Security table). In addition, three tables from DB3 770 are available for selection as nodes in the table graph: T2 771 (Events), T3 772 (Misc.), and T4 773 (History).

In some embodiments, if a user wishes to add a data table as a node into an existing or a new table graph, the user may drag a table and drop it into the table graph, at which time the table is inserted as a node in the table graph. In the example user interface 750, a user has dragged and dropped three tables as nodes into the table graph: node 751 (DB1.T1), node 752 (DB3.T2), and node 753 (DB3.T3). Edges may also be added between the tables. A user may also click on an edge and create indices on the tables and make further configurations as to the mapping between data fields in the nodes.

Figure 7 illustrates an example user interface 700 that displays a table graph that includes nodes that can be executed in parallel. In this example the displayed table graph 720 includes a total of eight nodes, several of which are configured to execute concurrently with certain other nodes, as discussed below.

A table graph that includes at least two nodes that are configured to be executed in parallel may be called a parallel table graph. In some embodiments, one or more nodes in a table graph may be included in a group so that searches related to the nodes in the group may be executed in parallel, making the searches even more efficient. For instance, in the example table graph 720, the nodes may be grouped automatically (and/or manually) into five different groups. In this example, Group 1 includes node A 721 and node B 722. Group 2 includes node C 723 and node E 725. Group 3 only includes one node, which is node D 724. Group 4

includes only one node, node F 726. Finally, group 5 includes two nodes, node G 727 and node H 728. In one embodiment, the nodes that belong to the same group may be executed in parallel because the search queries involving each of these nodes may be configured to be executed independently.

In some instances, there could be more than one way to partition a table graph into multiple groups. A table graph may be partitioned into several sub graphs in multiple ways. Therefore, besides automatically grouping several nodes together into one group, a user may choose to alter the grouping of nodes (e.g., an automatic grouping provided by the table graph generation software) for a parallel table graph for reasons of execution efficiency, data quality, and so forth. A user may utilize an interface such as the example user interface 700 and edit the existing grouping of nodes for parallel table graph execution. If, after the user's configuration, certain nodes assigned into the same group by the user cannot be executed in parallel, the example user interface 700 may display a warning message to the user and let the user know that the current assignment of nodes into a group would not result in a viable parallel table graph execution.

The following is an example of parallel table graph execution: Suppose a parallel table graph includes 3 tables/nodes including Table 1 that links a customer name to a customer ID, Table 2 that links a customer ID to a phone number, and Table 3 that links a customer ID to an address. In a search for a customer name, an initial search of Table 1 may be performed. However, one a customer ID corresponding to the customer name is determined, a parallel search of Tables 2 and 3 may be performed. Accordingly, in this example, a parallel table graph execution may be performed automatically on nodes corresponding to Tables 2 and 3.

Figure 8 is a flowchart depicting an illustrative process of executing a parallel table graph (e.g., a table graph with at least two nodes configured to execute at least partially concurrently). The process of Figure 8 may be performed by the data access system 252 and/or the API 248 in response to input from a data analyst, for example, such as part of a data analysis software platform. However, the process may be performed by other computing systems. Depending on the embodiment, the method of Figure 8 may include fewer or

additional blocks and the blocks may be performed in an order that is different than illustrated.

The process 900 begins at block 905, wherein a table graph is accessed. Accessing a table graph may include receiving information regarding the edges and nodes in the table graph and the data fields that may be involved in the edges and nodes.

The process 900 proceeds to block 910 and instructions regarding executing some nodes in the table graph in parallel are received. As discussed previously, the instructions to execute certain nodes in parallel may be received from a user and/or automatically determined. In some other embodiments, such instructions may be received from an application or a physical computing device that have instructions stored on it to execute a search query using a table graph. For example, an application that interacts with a table graph regularly, such as a program that periodically queries a table graph and reports updated results, may include instructions to query the table graph in parallel. A user or an administrator may set up such instructions as part of the application. Alternatively, the table graph system may automatically determine that particular nodes may be queried in parallel, and may automatically execute the particular nodes in parallel when a search query is received.

The process 900 then proceeds to block 915, and the order of executing the nodes in the table graph are determined. Depending on the embodiments, the determination may be made automatically based on the dependencies of the search queries and/or other factors, such as expected time for executing a query, the complexity of a query, records of past search query execution duration, amount of data potentially involved in a search query/or in a node, database usage statistics, search demand predictions, and so forth. The factors may further be weighted in order to plan the search queries in a cost-effective way. For example, queries that may be repeated periodically can be scheduled to take place at certain times when the demand on a database server is low. In another example, nodes involved in search queries that, according to statistics, may take a long time to finish, may be executed with more priority than other search nodes involving other nodes, even if both could be executed in parallel.

In some other embodiments, the determination of the order of executing the nodes may also be made by a user, who may arrange the order of executing the nodes by assigning certain nodes into the same or various different execution stages and/or groups. Also, depending on the embodiment, the determination may be made by a combination of user input and automatic determination. A user may modify the automatic parallel table graph execution arrangement. A user's parallel table graph execution arrangement may undergo sanity checks performed automatically or as part of a program.

The process 900 then proceeds to block 920 and at least some of the nodes in the table graph are executed in parallel according to the determined order. In some embodiments, a user interface may also be executed and created to the user so that the user may monitor the execution of the table graph as it happens. In some other embodiments, performance statistics may be gathered for the table graph search in order to further optimize future parallel table graph searches.

Implementation Mechanisms

According to one embodiment, the techniques described herein are implemented by one or more special-purpose computing devices. The special-purpose computing devices may be hard-wired to perform the techniques, or may include digital electronic devices such as one or more application-specific integrated circuits (ASICs) or field programmable gate arrays (FPGAs) that are persistently programmed to perform the techniques, or may include one or more general purpose hardware processors programmed to perform the techniques pursuant to program instructions in firmware, memory, other storage, or a combination. Such special-purpose computing devices may also combine custom hard-wired logic, ASICs, or FPGAs with custom programming to accomplish the techniques. The special-purpose computing devices may be desktop computer systems, server computer systems, portable computer systems, handheld devices, networking devices or any other device or combination of devices that incorporate hard-wired and/or program logic to implement the techniques.

Computing device(s) are generally controlled and coordinated by operating system software, such as iOS, Android, Chrome OS, Windows XP, Windows Vista, Windows 7, Windows 8, Windows Server, Windows CE, Unix, Linux, SunOS, Solaris, iOS, Blackberry OS, VxWorks, or other compatible operating systems. In other embodiments, the computing

device may be controlled by a proprietary operating system. Conventional operating systems control and schedule computer processes for execution, perform memory management, provide file system, networking, I/O services, and provide a user interface functionality, such as a graphical user interface ("GUI"), among other things.

For example, Figure 9 is a block diagram that illustrates a computer system 800 upon which the processed discussed herein may be implemented. For example, a table graph generation user interface may be generated and displayed to a user by a first computer system 800, while a search query using one or more table graphs may be executed by another computer system 800 (or possibly the same computer system in some embodiments). Furthermore the data sources may each include any portion of the components and functionality discussed with reference to the computer system 800.

The example computer system 800 includes a bus 802 or other communication mechanism for communicating information, and a hardware processor, or multiple processors, 804 coupled with bus 802 for processing information. Hardware processor(s) 804 may be, for example, one or more general purpose microprocessors.

Computer system 800 also includes a main memory 806, such as a random access memory (RAM), cache and/or other dynamic storage devices, coupled to bus 802 for storing information and instructions to be executed by processor 804. Main memory 806 also may be used for storing temporary variables or other intermediate information during execution of instructions to be executed by processor 804. Such instructions, when stored in storage media accessible to processor 804, render computer system 800 into a special-purpose machine that is customized to perform the operations specified in the instructions.

Computer system 800 further includes a read only memory (ROM) 808 or other static storage device coupled to bus 802 for storing static information and instructions for processor 804. A storage device 810, such as a magnetic disk, optical disk, or USB thumb drive (Flash drive), etc., is provided and coupled to bus 802 for storing information and instructions.

Computer system 800 may be coupled via bus 802 to a display 812, such as a cathode ray tube (CRT) or LCD display (or touch screen), for displaying information to a computer user. An input device 814, including alphanumeric and other keys, is coupled to bus 802 for communicating information and command selections to processor 804. Another type of user

input device is cursor control 816, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 804 and for controlling cursor movement on display 812. This input device typically has two degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), that allows the device to specify positions in a plane. In some embodiments, the same direction information and command selections as cursor control may be implemented via receiving touches on a touch screen without a cursor.

Computing system 800 may include a user interface module to implement a GUI that may be stored in a mass storage device as executable software codes that are executed by the computing device(s). This and other modules may include, by way of example, components, such as software components, object-oriented software components, class components and task components, processes, functions, attributes, procedures, subroutines, segments of program code, drivers, firmware, microcode, circuitry, data, databases, data structures, tables, arrays, and variables.

In general, the word "module," as used herein, refers to logic embodied in hardware or firmware, or to a collection of software instructions, possibly having entry and exit points, written in a programming language, such as, for example, Java, Lua, C or C++. A software module may be compiled and linked into an executable program, installed in a dynamic link library, or may be written in an interpreted programming language such as, for example, BASIC, Perl, or Python. It will be appreciated that software modules may be callable from other modules or from themselves, and/or may be invoked in response to detected events or interrupts. Software modules configured for execution on computing devices may be provided on a computer readable medium, such as a compact disc, digital video disc, flash drive, magnetic disc, or any other tangible medium, or as a digital download (and may be originally stored in a compressed or installable format that requires installation, decompression or decryption prior to execution). Such software code may be stored, partially or fully, on a memory device of the executing computing device, for execution by the computing device. Software instructions may be embedded in firmware, such as an EPROM. It will be further appreciated that hardware modules may be comprised of connected logic units, such as gates and flip-flops, and/or may be comprised of programmable units, such as

programmable gate arrays or processors. The modules or computing device functionality described herein are preferably implemented as software modules, but may be represented in hardware or firmware. Generally, the modules described herein refer to logical modules that may be combined with other modules or divided into sub-modules despite their physical organization or storage

Computer system 800 may implement the techniques described herein using customized hard-wired logic, one or more ASICs or FPGAs, firmware and/or program logic which in combination with the computer system causes or programs computer system 800 to be a special-purpose machine. According to one embodiment, the techniques herein are performed by computer system 800 in response to processor(s) 804 executing one or more sequences of one or more instructions contained in main memory 806. Such instructions may be read into main memory 806 from another storage medium, such as storage device 810. Execution of the sequences of instructions contained in main memory 806 causes processor(s) 804 to perform the process steps described herein. In alternative embodiments, hard-wired circuitry may be used in place of or in combination with software instructions.

The term "non-transitory media," and similar terms, as used herein refers to any media that store data and/or instructions that cause a machine to operate in a specific fashion. Such non-transitory media may comprise non-volatile media and/or volatile media. Non-volatile media includes, for example, optical or magnetic disks, such as storage device 810. Volatile media includes dynamic memory, such as main memory 806. Common forms of non-transitory media include, for example, a floppy disk, a flexible disk, hard disk, solid state drive, magnetic tape, or any other magnetic data storage medium, a CD-ROM, any other optical data storage medium, any physical medium with patterns of holes, a RAM, a PROM, and EPROM, a FLASH-EPROM, NVRAM, any other memory chip or cartridge, and networked versions of the same.

Non-transitory media is distinct from but may be used in conjunction with transmission media. Transmission media participates in transferring information between nontransitory media. For example, transmission media includes coaxial cables, copper wire and fiber optics, including the wires that comprise bus 802. Transmission media can also take

the form of acoustic or light waves, such as those generated during radio-wave and infra-red data communications.

Various forms of media may be involved in carrying one or more sequences of one or more instructions to processor 804 for execution. For example, the instructions may initially be carried on a magnetic disk or solid state drive of a remote computer. The remote computer can load the instructions into its dynamic memory and send the instructions over a telephone line using a modem. A modem local to computer system 800 can receive the data on the telephone line and use an infra-red transmitter to convert the data to an infra-red signal. An infra-red detector can receive the data carried in the infra-red signal and appropriate circuitry can place the data on bus 802. Bus 802 carries the data to main memory 806, from which processor 804 retrieves and executes the instructions. The instructions received by main memory 806 may retrieves and executes the instructions. The instructions received by main memory 806 may optionally be stored on storage device 810 either before or after execution by processor 804.

Computer system 800 also includes a communication interface 818 coupled to bus 802. Communication interface 818 provides a two-way data communication coupling to a network link 820 that is connected to a local network 822. For example, communication interface 818 may be an integrated services digital network (ISDN) card, cable modem, satellite modem, or a modem to provide a data communication connection to a corresponding type of telephone line. As another example, communication interface 818 may be a local area network (LAN) card to provide a data communication connection to a compatible LAN (or WAN component to communicated with a WAN). Wireless links may also be implemented. In any such implementation, communication interface 818 sends and receives electrical, electromagnetic or optical signals that carry digital data streams representing various types of information.

Network link 820 typically provides data communication through one or more networks to other data devices. For example, network link 820 may provide a connection through local network 822 to a host computer 824 or to data equipment operated by an Internet Service Provider (ISP) 826. ISP 826 in turn provides data communication services through the world wide packet data communication network now commonly referred to as

the "Internet" 828. Local network 822 and Internet 828 both use electrical, electromagnetic or optical signals that carry digital data streams. The signals through the various networks and the signals on network link 820 and through communication interface 818, which carry the digital data to and from computer system 800, are example forms of transmission media.

Computer system 800 can send messages and receive data, including program code, through the network(s), network link 820 and communication interface 818. In the Internet example, a server 830 might transmit a requested code for an application program through Internet 828, ISP 826, local network 822 and communication interface 818.

The received code may be executed by processor 804 as it is received, and/or stored in storage device 810, or other non-volatile storage for later execution.

Each of the processes, methods, and algorithms described in the preceding sections may be embodied in, and fully or partially automated by, code modules executed by one or more computer systems or computer processors comprising computer hardware. The processes and algorithms may be implemented partially or wholly in application-specific circuitry.

The various features and processes described above may be used independently of one another, or may be combined in various ways. All possible combinations and subcombinations are intended to fall within the scope of this disclosure. In addition, certain method or process blocks may be omitted in some implementations. The methods and processes described herein are also not limited to any particular sequence, and the blocks or states relating thereto can be performed in other sequences that are appropriate. For example, described blocks or states may be performed in an order other than that specifically disclosed, or multiple blocks or states may be combined in a single block or state. The example blocks or states may be performed in serial, in parallel, or in some other manner. Blocks or states may be added to or removed from the disclosed example embodiments. The example systems and components described herein may be configured differently than described. For example, elements may be added to, removed from, or rearranged compared to the disclosed example embodiments.

Conditional language, such as, among others, "can," "could," "might," or "may," unless specifically stated otherwise, or otherwise understood within the context as used, is

generally intended to convey that certain embodiments include, while other embodiments do not include, certain features, elements and/or steps. Thus, such conditional language is not generally intended to imply that features, elements and/or steps are in any way required for one or more embodiments or that one or more embodiments necessarily include logic for deciding, with or without user input or prompting, whether these features, elements and/or steps are included or are to be performed in any particular embodiment.

The term "comprising" as used herein should be given an inclusive rather than exclusive interpretation. For example, a general purpose computer comprising one or more processors should not be interpreted as excluding other computer components, and may possibly include such components as memory, input/output devices, and/or network interfaces, among others.

Any process descriptions, elements, or blocks in the flow diagrams described herein and/or depicted in the attached figures should be understood as potentially representing modules, segments, or portions of code which include one or more executable instructions for implementing specific logical functions or steps in the process. Alternate implementations are included within the scope of the embodiments described herein in which elements or functions may be deleted, executed out of order from that shown or discussed, including substantially concurrently or in reverse order, depending on the functionality involved, as would be understood by those skilled in the art.

It should be emphasized that many variations and modifications may be made to the above-described embodiments, the elements of which are to be understood as being among other acceptable examples. All such modifications and variations are intended to be included herein within the scope of this disclosure. The foregoing description details certain embodiments of the invention. It will be appreciated, however, that no matter how detailed the foregoing appears in text, the invention can be practiced in many ways. As is also stated above, it should be noted that the use of particular terminology when describing certain features or aspects of the invention should not be taken to imply that the terminology is being re-defined herein to be restricted to including any specific characteristics of the features or aspects of the invention with which that terminology is associated. The scope of the appended

claims should not be limited by the specific embodiments set forth, but should be given the broadest interpretation consistent with the teachings of the description as a whole.

WHAT IS CLAIMED IS:

1. A search system configured to execute a search query, the search system comprising:

one or more computing devices having hardware processors configured to execute instructions in order to:

access a first data store of a first type storing at least a first data table;

access a second data store of a second type different than the first type, the second data store storing at least a second data table;

execute a table graph having a first node associated with the first data table, a second node associated with the second data table, and a link between the first node and the second node, the link indicating a first field of the first data table that is associated with a second field of the second data table, said executing comprising:

looking up first information associated with a search query in the first data store;

looking up second information associated with the first information in the second data store;

providing the second information for display or processing by the search system.

2. The search system of Claim 1, wherein at least one of the first field and the second field are full-text searchable.

3. The search system of Claim 1 or 2, wherein the link between the first node and the second node is a bi-directional link.

4. The search system of any one of Claims 1 to 3, wherein the first type and the second type each include one or more of a relational data store, an object-oriented data store, a proprietary data store, a file-based data store, a hierarchical data store, a network data store, and an elastic-search data store.

5. The search system of any one of Claims 1 to 4, wherein the table graph is executed via an application programming interface configured to format the second

information as one or more data objects, wherein the data objects further comprise attributes and values defined by a user.

6. The search system of any one of Claims 1 to 5, wherein the search query includes one or more regular expression rules.

7. The search system of any one of Claims 1 to 6, wherein the table graph further comprises a third node associated with a third data table and wherein executing the table graph further comprises concurrently looking up the first information in the first data table and looking up a third information in the third data table.

8. The search system of Claim 7, wherein the lookups in the first data table and the third data tables are executed concurrently in response to the one or more computing devices automatically determining that the lookups are suitable for concurrent execution based on at least one of: query dependency, query complexity, history of query execution duration, usage statistics of the first, second, and/or third data stores, and/or predicted usage demand of the first, second, and third data stores.

9. A table graph system configured to electronically communicate with at least one data store, the table graph system comprising:

one or more physical computing devices having hardware processors configured to execution instructions in order to:

receive instructions to add a first data table as a first node in a table graph;

receive instructions to add a second data table as a second node in the table graph, wherein the first data table and the second data table are of different types;

upon receiving instructions to add a relationship between the first and second nodes, add the relationship, wherein the relationship indicates a first field of the first data table is to be associated with a second field of the second data table,

wherein the table graph is executable in order to access data stored in the first data table and subsequently data in the second data table based at least partly on the data stored in the first data table.

10. The system of Claim 9, wherein the one or more physical computing devices are further configured to:

receive instructions to add a third node linked with the first node and/or the second node;

receive instructions to indicate the first node and the third node as concurrently processable, wherein the first node and the third node are executable concurrently in response to execution of the table graph.

11. The system of Claim 9, wherein the one or more physical computing devices are further configured to:

receive instructions to add a third node linked with the first node and/or the second node;

receive instructions to execute at least two of the nodes concurrently;

automatically determine an order of executing the at least two of the nodes concurrently.

12. The system of any one of Claims 9 to 11, wherein the one or more physical computing devices are further configured to automatically add one or more relationships between the first and second nodes.

13. The system of any one of Claims 9 to 12, wherein the one or more physical computing devices are further configured to receive instructions to change the relationship between the first and second nodes from a one-directional link to a bi-directional link.

14. The system of any one of Claims 9 to 13, wherein the first data table and the second data table are located remotely from each other.

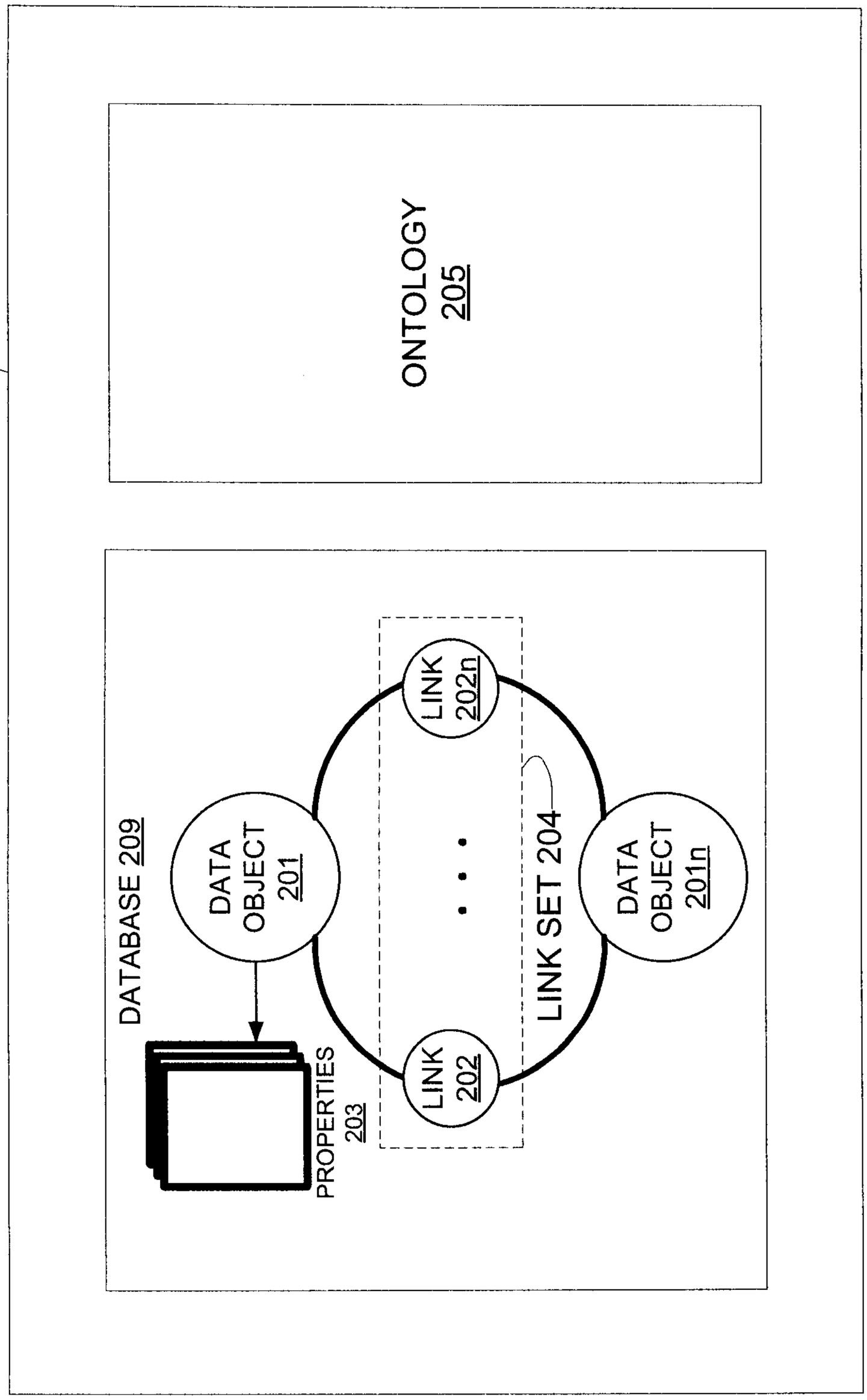

Database System 210
ONTOLOGY 205
DATABASE 209
DATA OBJECT 201
PROPERTIES 203
LINK 202
LINK SET 204
DATA OBJECT 201n
LINK 202n

FIG. 1

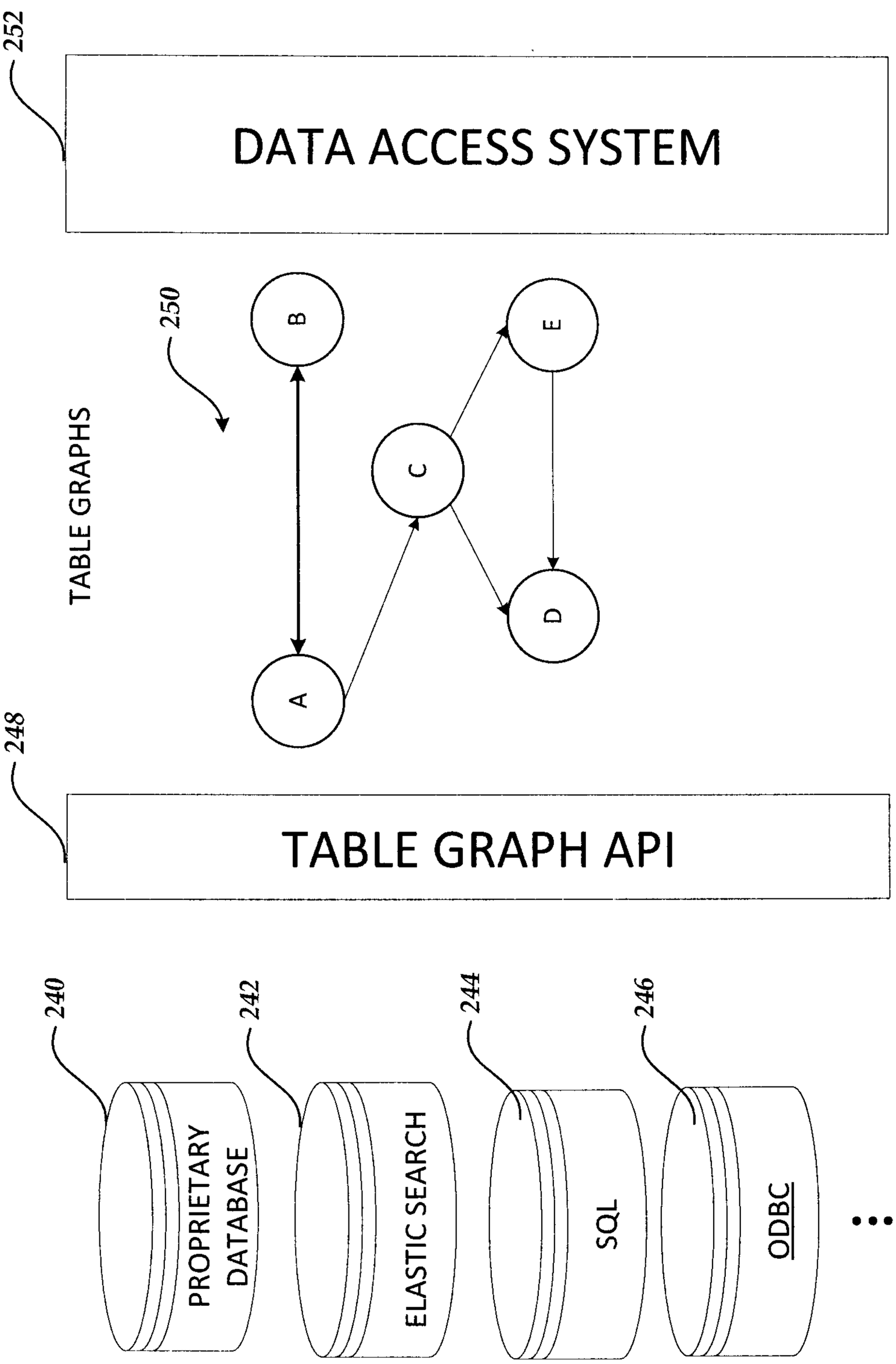

252
DATA ACCESS SYSTEM
254
USERS
256
APPLICATIONS
250
TABLE GRAPHS
A
B
C
D
E
248
TABLE GRAPH API
240
242
244
246
PROPRIETARY DATABASE
ELASTIC SEARCH
SQL
ODBC
...

FIG. 2

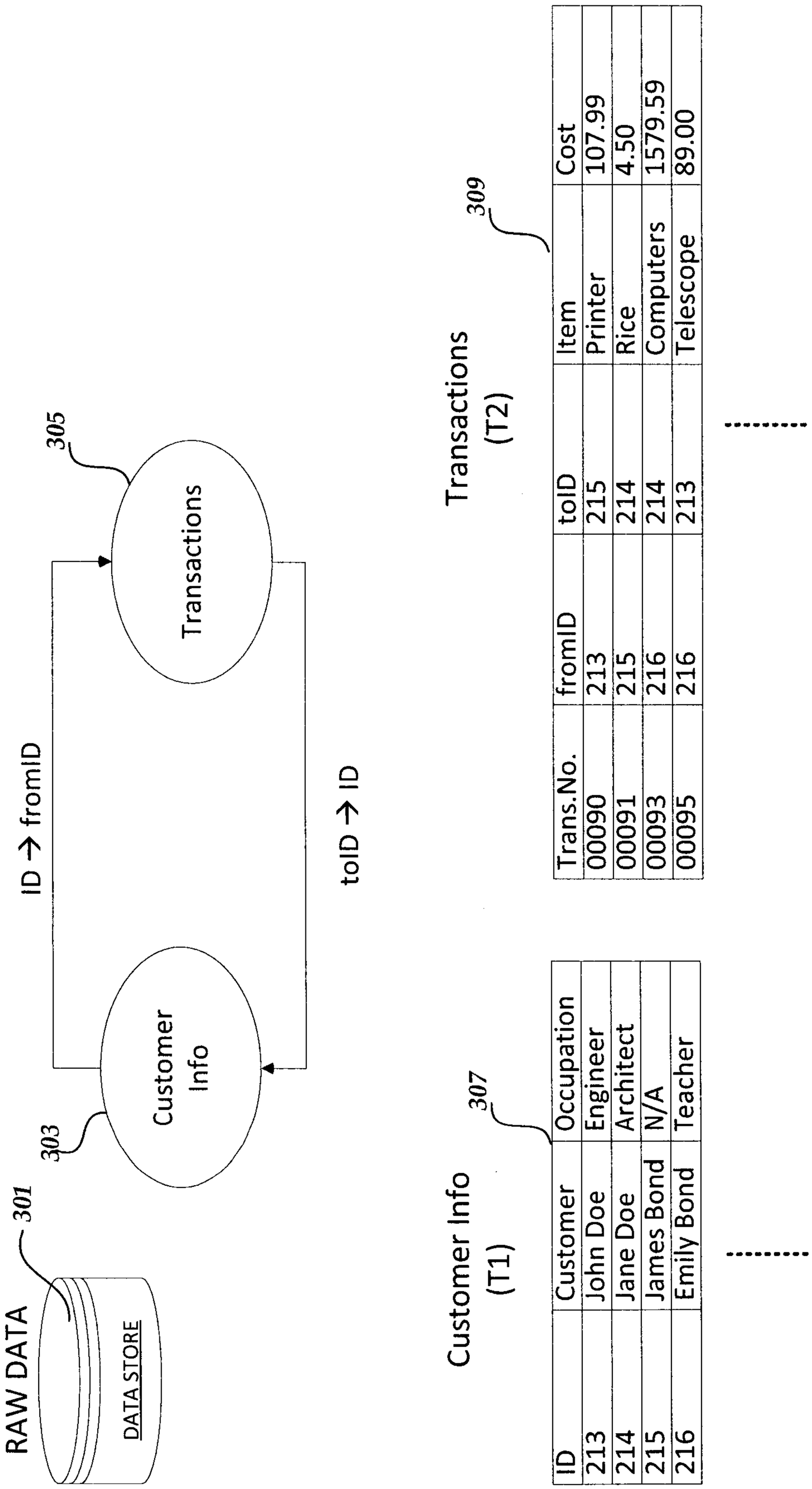
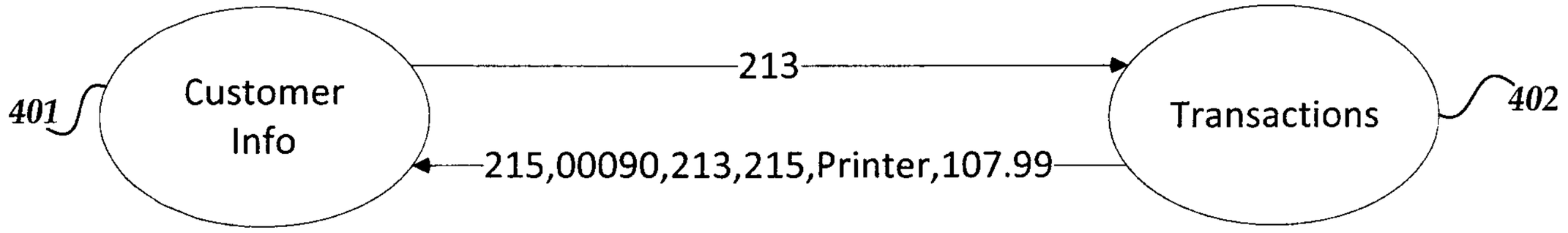

RAW DATA
301
DATA STORE
303
Customer Info
305
Transactions
ID → fromID
toID → ID
Customer Info (T1)
307
Customer | Occupation
John Doe | Engineer
Jane Doe | Architect
James Bond | N/A
Emily Bond | Teacher
ID
213
214
215
216
Transactions (T2)
309
Trans.No. | fromID | toID | Cost
00090 | 213 | 215 | 107.99
00091 | 215 | 214 | 4.50
00093 | 216 | 214 | 1579.59
00095 | 216 | 213 | 89.00

FIG. 3

Find Transactions Associated with customer ID 213 (John Doe) and Entities involved in those transactions



Customer Info (T1)

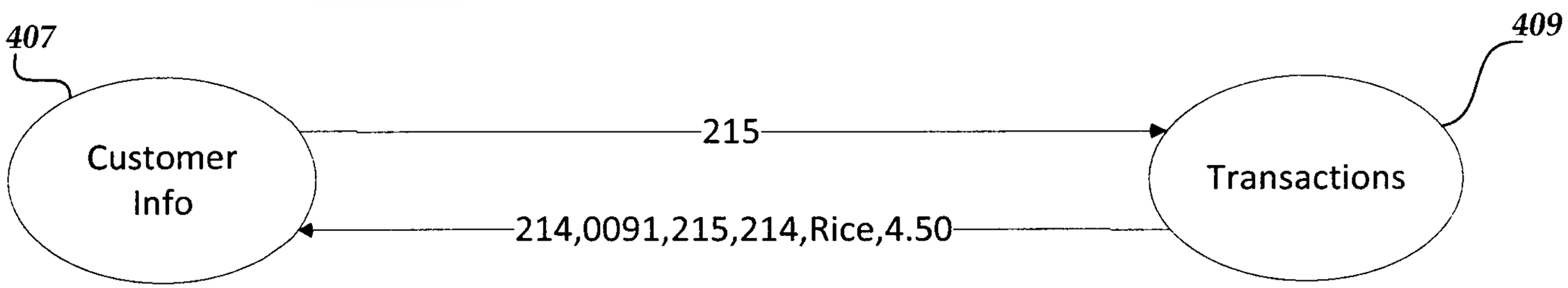
307

ID	Customer	Occupation
213	John Doe	Engineer
214	Jane Doe	Architect
215	James Bond	N/A
216	Emily Bond	Teacher

Transactions (T2)

309

Trans.No.	fromID	toID	Item	Cost
00090	213	215	Printer	107.99
00091	215	214	Rice	4.50
00093	216	214	Computers	1579.59
00095	216	213	Telescope	89.00



Customer Info (T1)

307

ID	Customer	Occupation
213	John Doe	Engineer
214	Jane Doe	Architect
215	James Bond	N/A
216	Emily Bond	Teacher

Transactions (T2)

309

Trans.No.	fromID	toID	Item	Cost
00090	213	215	Printer	107.99
00091	215	214	Rice	4.50
00093	216	214	Computers	1579.59
00095	216	213	Telescope	89.00

FIG. 4A

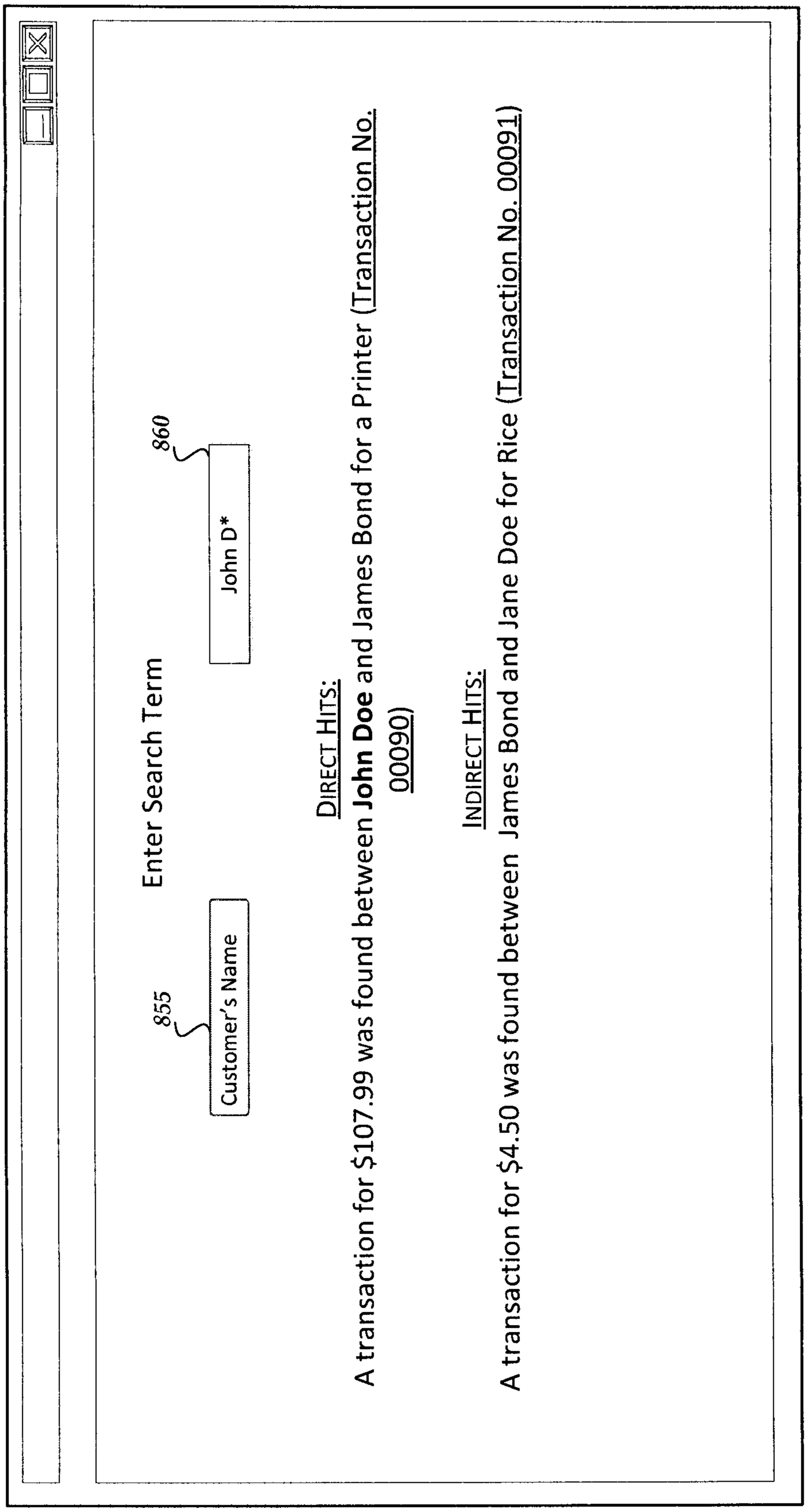

850
Enter Search Term
860
John D*
855
Customer's Name
DIRECT HITS:
A transaction for $107.99 was found between John Doe and James Bond for a Printer (Transaction No. 00090)
INDIRECT HITS:
A transaction for $4.50 was found between James Bond and Jane Doe for Rice (Transaction No. 00091)

FIG. 4B

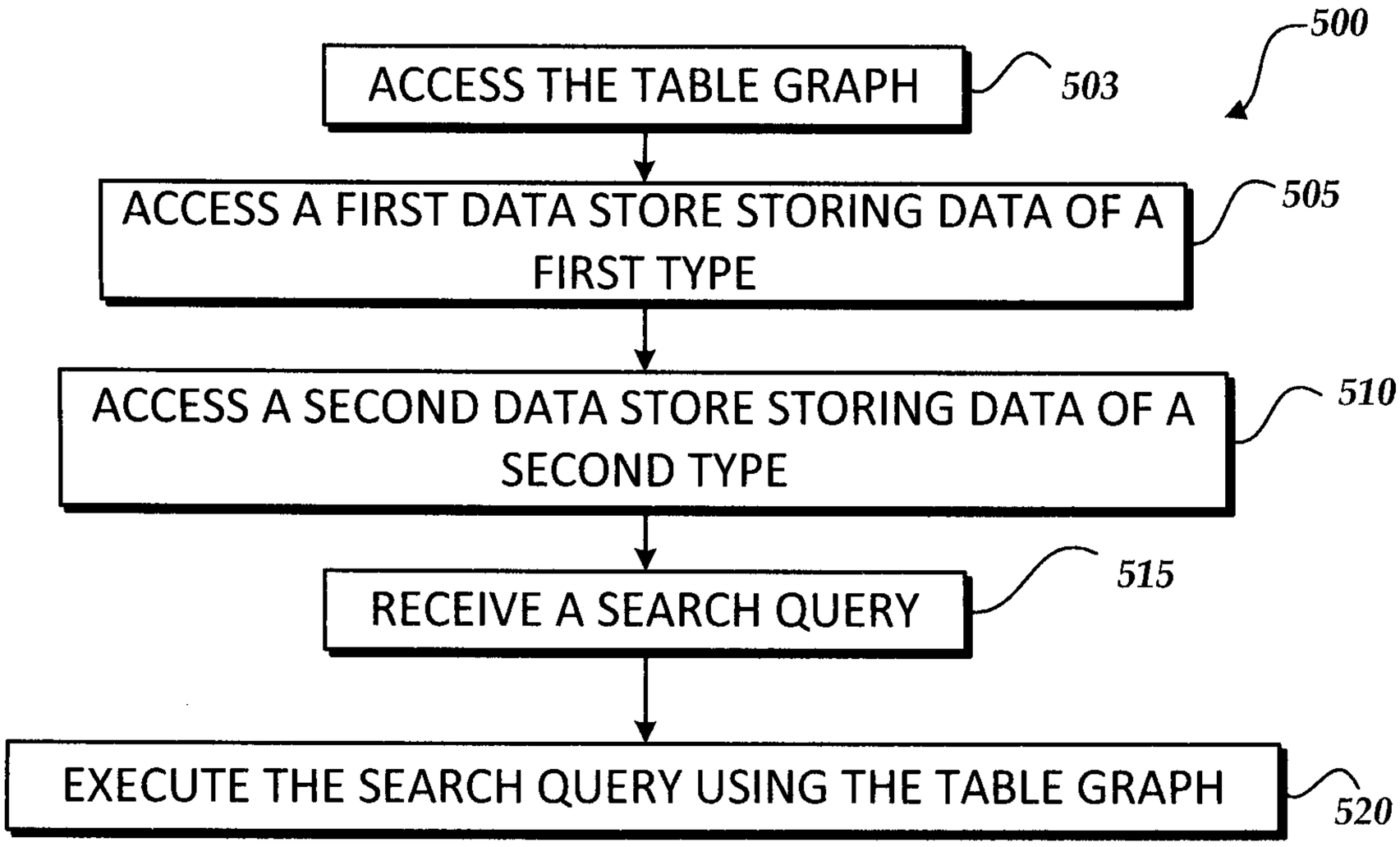

500
ACCESS THE TABLE GRAPH
503
ACCESS A FIRST DATA STORE STORING DATA OF A FIRST TYPE
505
ACCESS A SECOND DATA STORE STORING DATA OF A SECOND TYPE
510
RECEIVE A SEARCH QUERY
515
EXECUTE THE SEARCH QUERY USING THE TABLE GRAPH
520

FIG. 5A

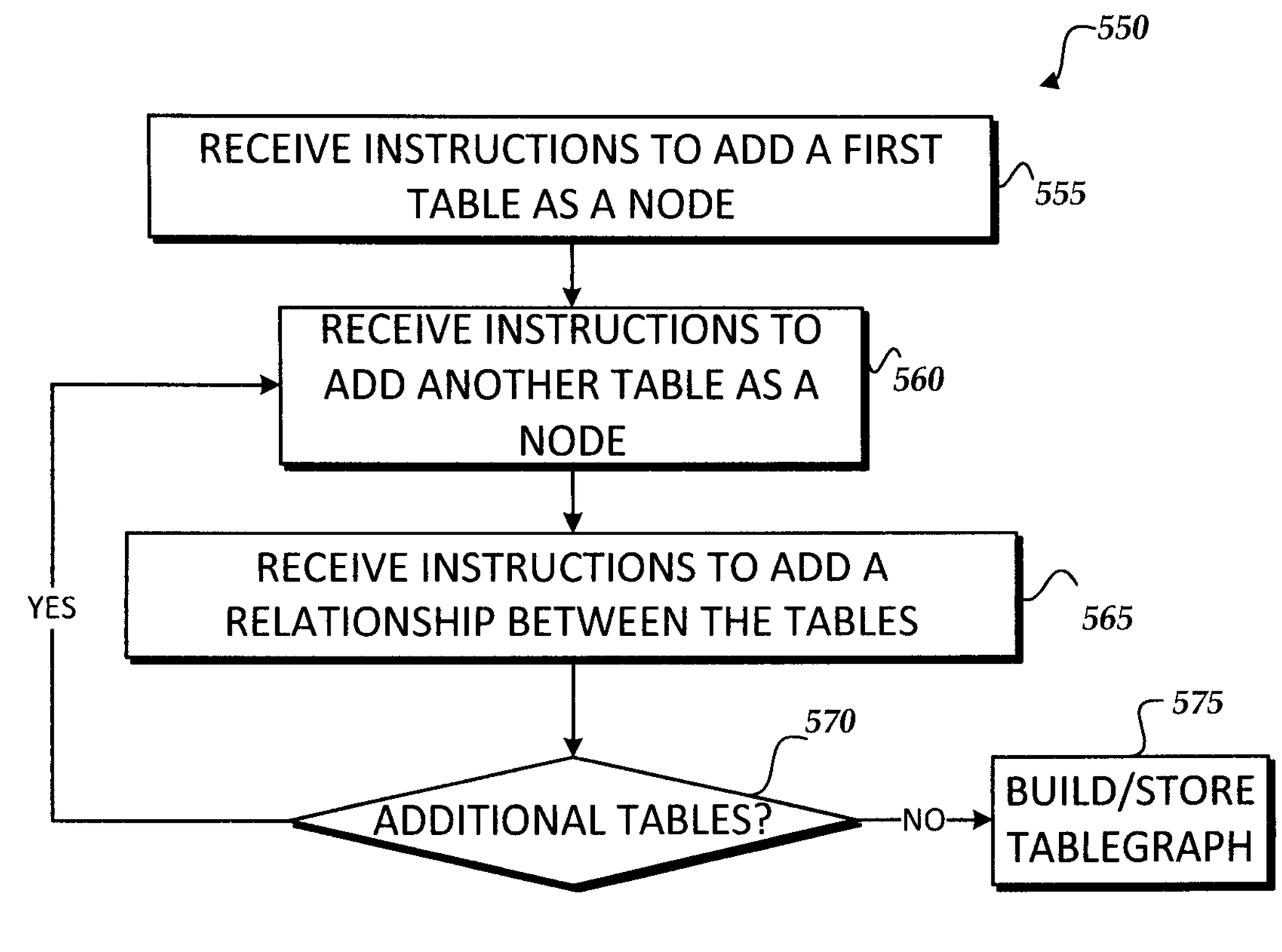

550
RECEIVE INSTRUCTIONS TO ADD A FIRST TABLE AS A NODE
555
RECEIVE INSTRUCTIONS TO ADD ANOTHER TABLE AS A NODE
560
RECEIVE INSTRUCTIONS TO ADD A RELATIONSHIP BETWEEN THE TABLES
565
YES
570
ADDITIONAL TABLES?
NO
575
BUILD/STORE TABLEGRAPH

FIG. 5B

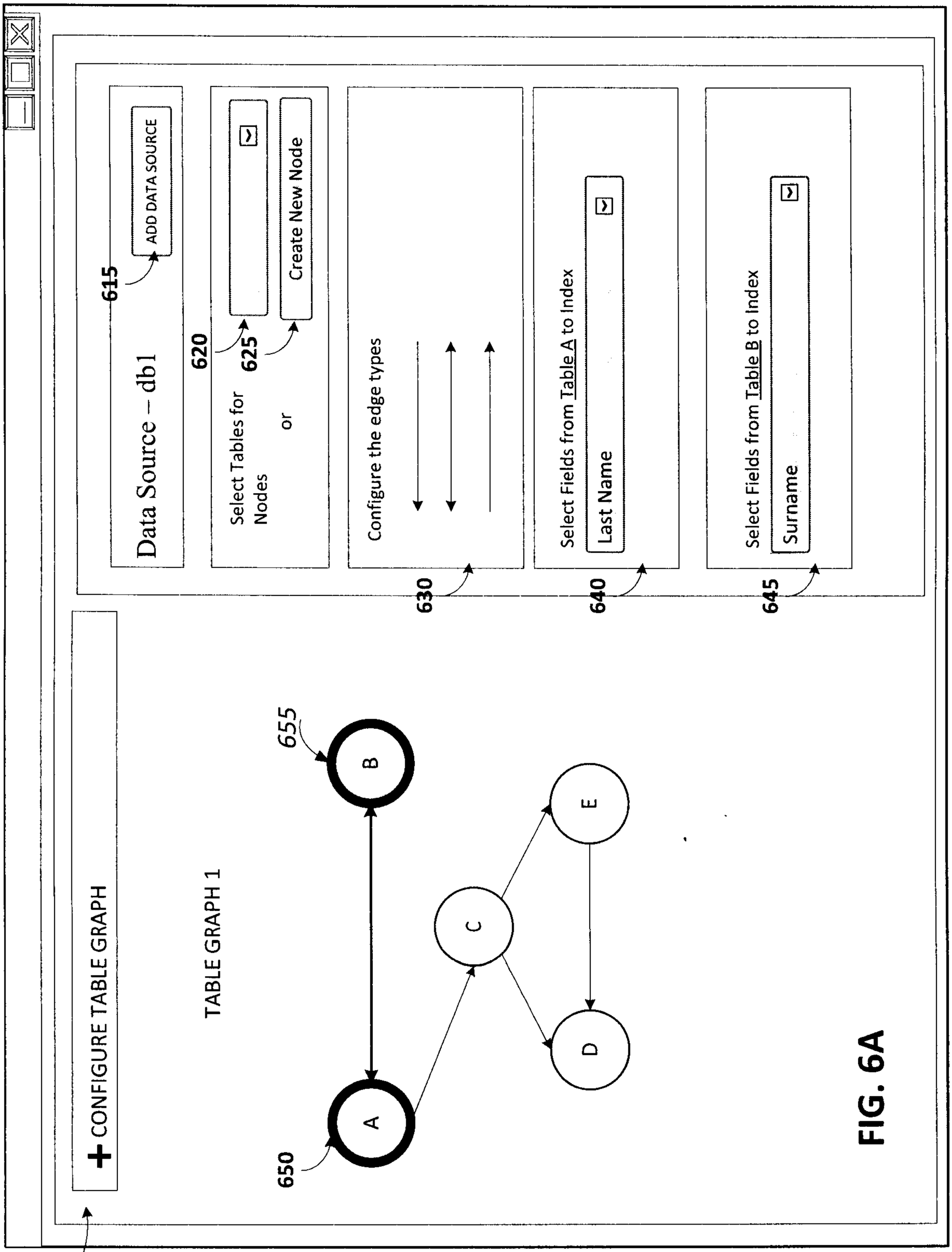

600
610
CONFIGURE TABLE GRAPH
615
ADD DATA SOURCE
Data Source – db1
620
Select Tables for Nodes
or
625
Create New Node
630
Configure the edge types
640
Select Fields from Table A to Index
Last Name
645
Select Fields from Table B to Index
Surname
TABLE GRAPH 1
650
655
A
B
C
D
E

FIG. 6A

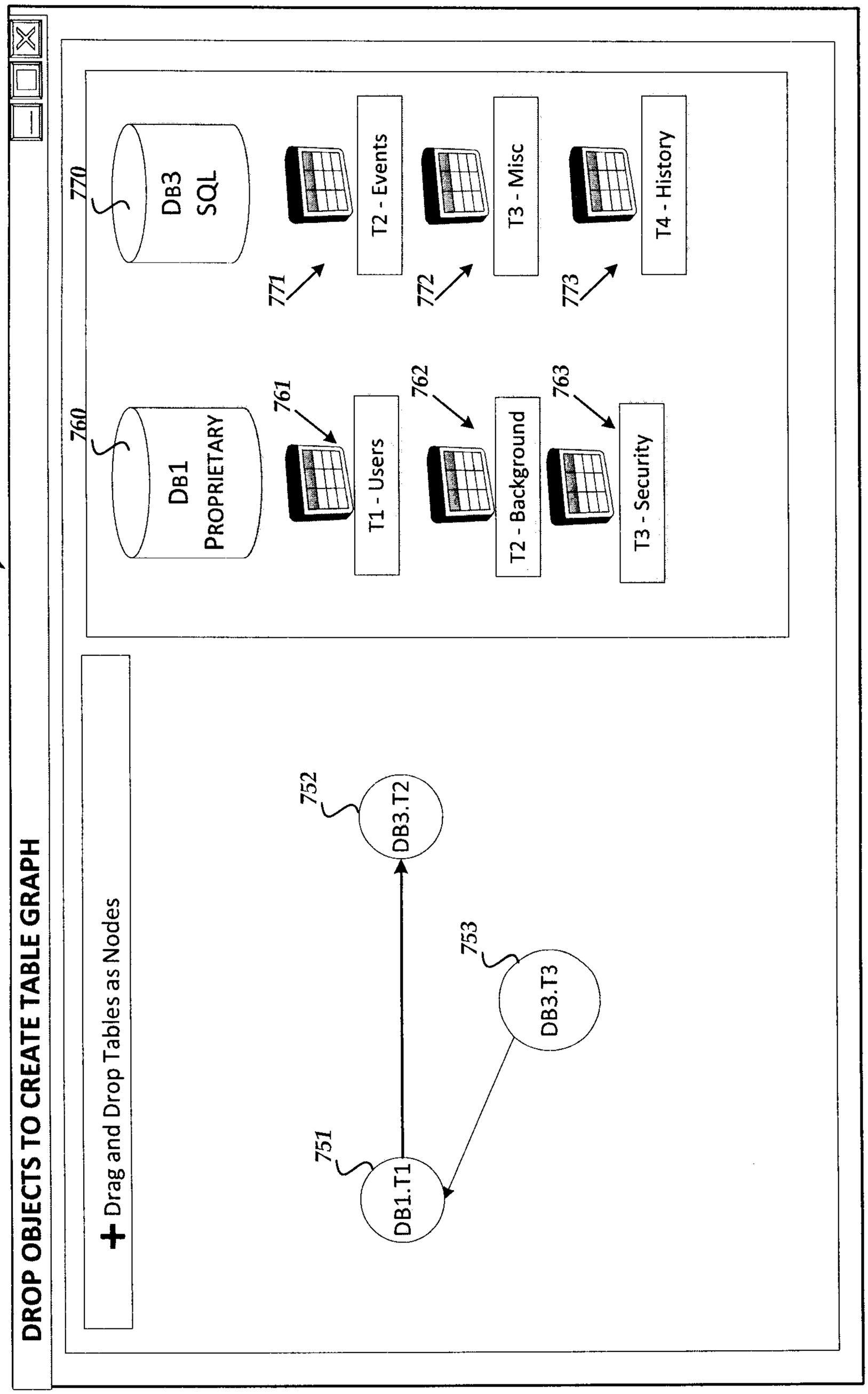

750
DROP OBJECTS TO CREATE TABLE GRAPH
+ Drag and Drop Tables as Nodes
760
DB1
PROPRIETARY
761
T1 - Users
762
T2 - Background
763
T3 - Security
770
DB3
SQL
771
T2 - Events
772
T3 - Misc
773
T4 - History
751
DB1.T1
752
DB3.T2
753
DB3.T3

FIG. 6B

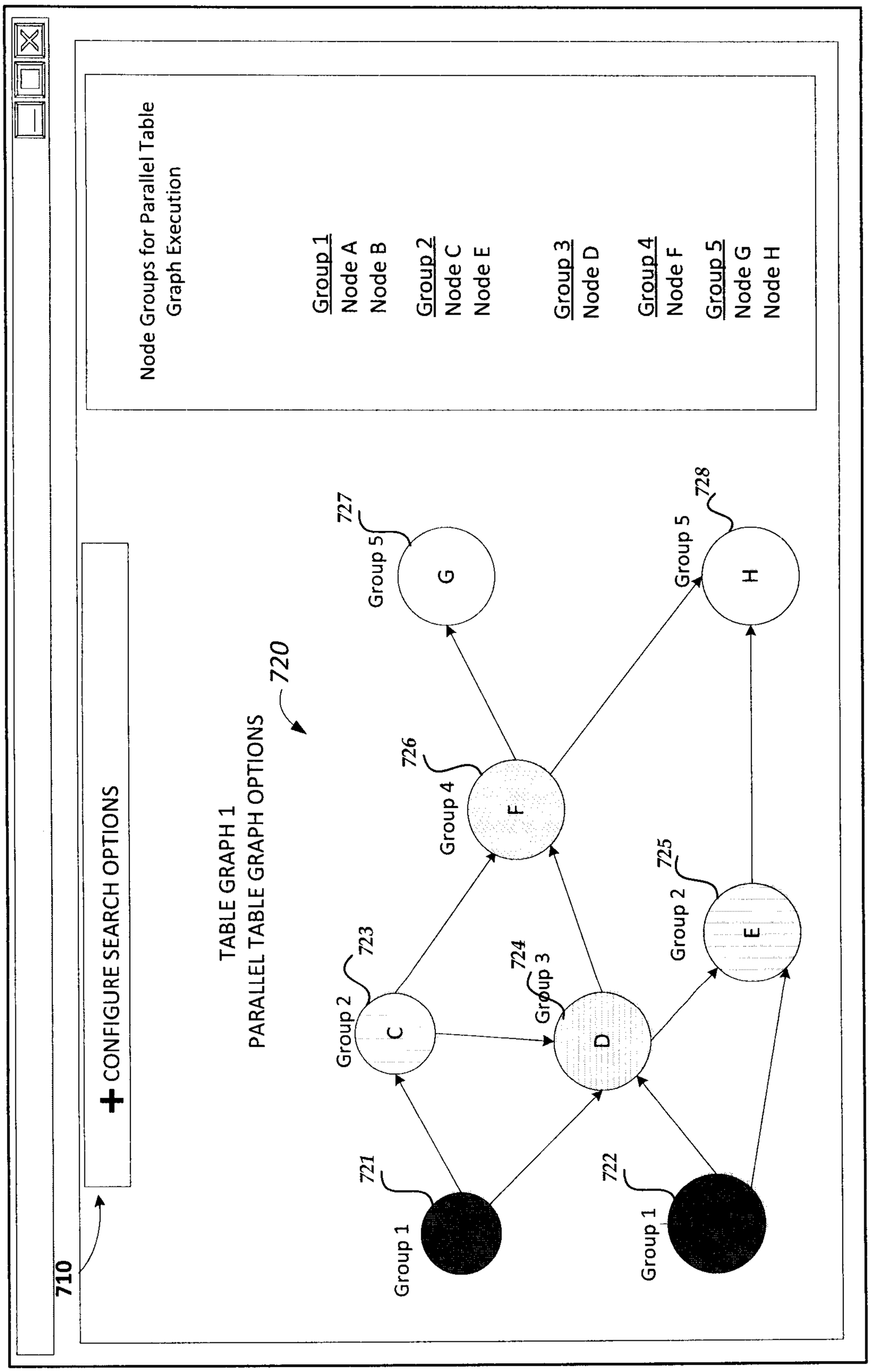

700
710
CONFIGURE SEARCH OPTIONS
720
TABLE GRAPH 1
PARALLEL TABLE GRAPH OPTIONS
Node Groups for Parallel Table
Graph Execution
Group 1
Node A
Node B
Group 2
Node C
Node E
Group 3
Node D
Group 4
Node F
Group 5
Node G
Node H
721
722
723
724
725
726
727
728
Group 1
Group 2
Group 3
Group 4
Group 5
A
B
C
D
E
F
G
H

FIG. 7

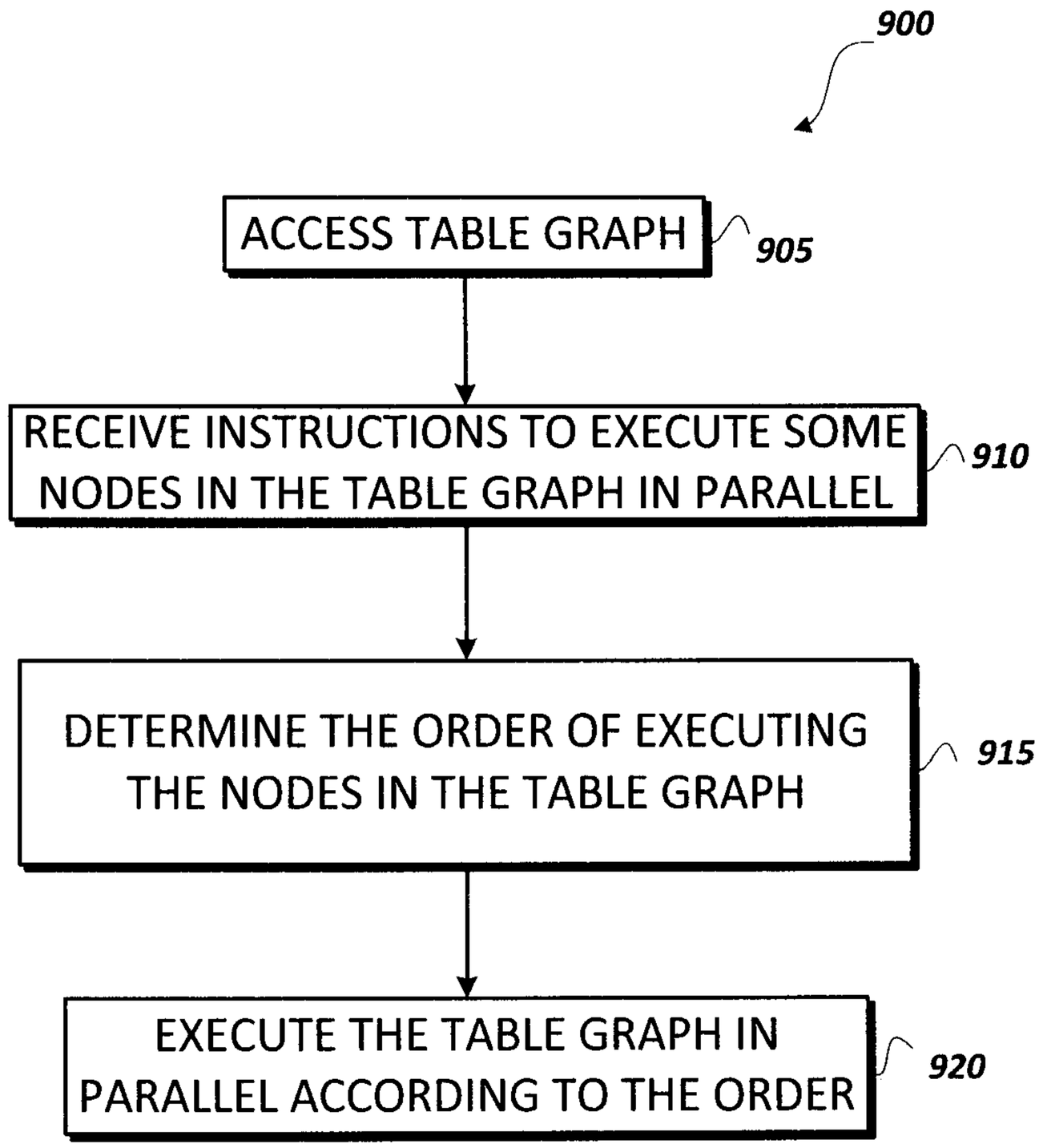

900
ACCESS TABLE GRAPH
905
RECEIVE INSTRUCTIONS TO EXECUTE SOME NODES IN THE TABLE GRAPH IN PARALLEL
910
DETERMINE THE ORDER OF EXECUTING THE NODES IN THE TABLE GRAPH
915
EXECUTE THE TABLE GRAPH IN PARALLEL ACCORDING TO THE ORDER
920

FIG. 8

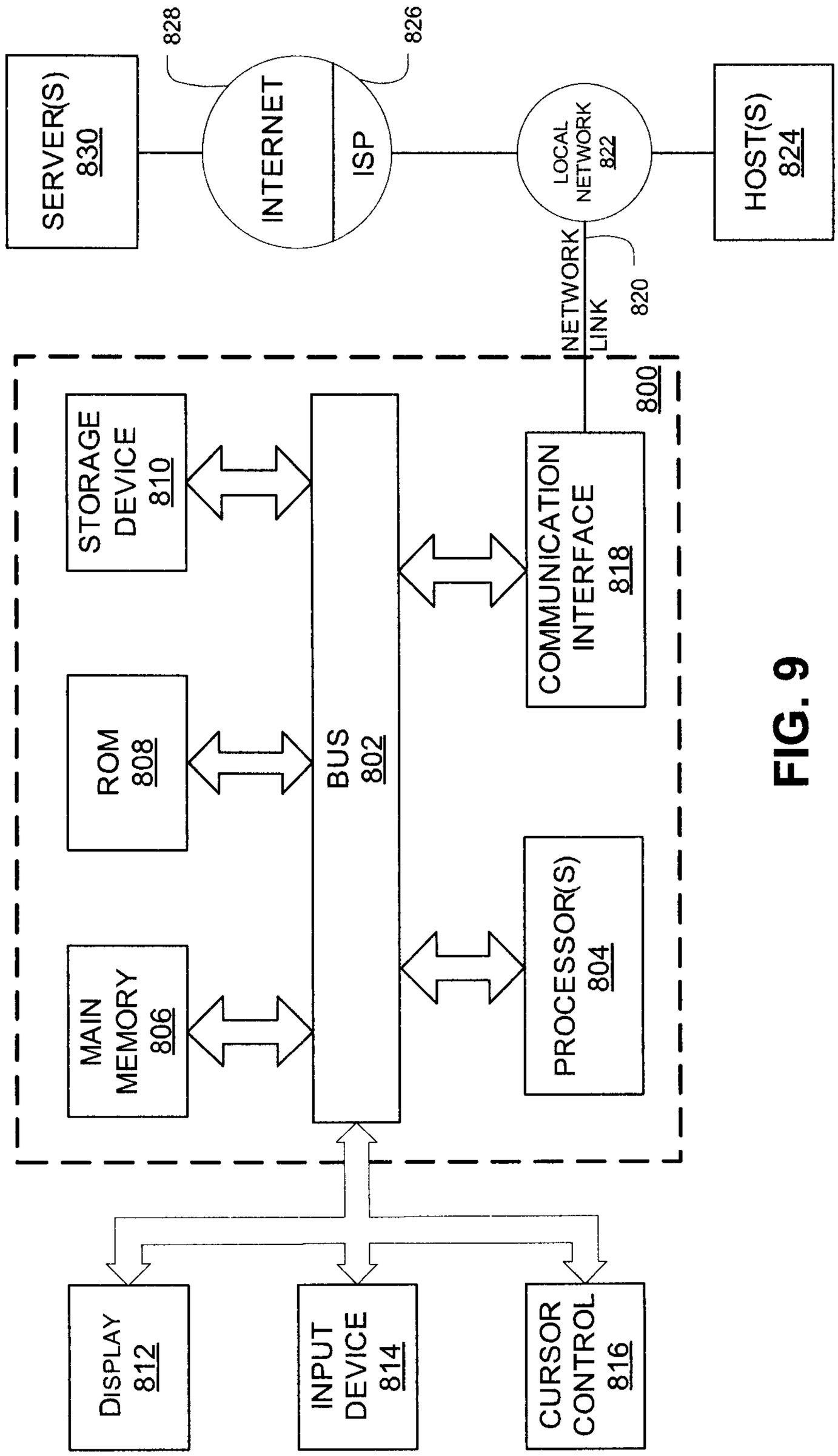
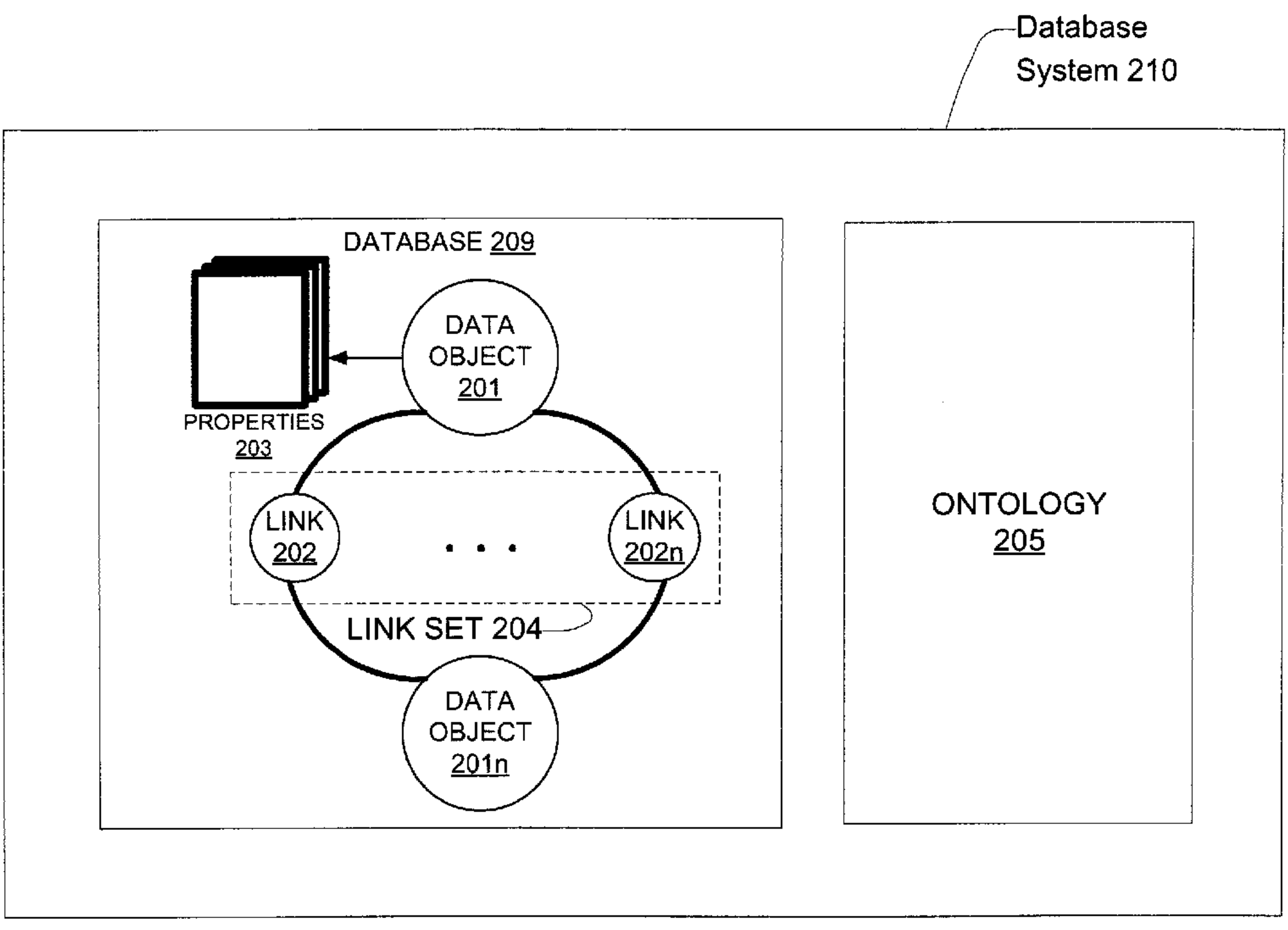

SERVER(S)
830
828
INTERNET
ISP
826
LOCAL
NETWORK
822
HOST(S)
824
NETWORK
LINK
820
800
STORAGE
DEVICE
810
ROM
808
MAIN
MEMORY
806
BUS
802
COMMUNICATION
INTERFACE
818
PROCESSOR(S)
804
DISPLAY
812
INPUT
DEVICE
814
CURSOR
CONTROL
816

FIG. 9


Database
System 210
DATABASE 209
DATA
OBJECT
201
PROPERTIES
203
LINK
202
LINK
202n
LINK SET 204
DATA
OBJECT
201n
ONTOLOGY
205