US Patent & Trademark Office
Patent Public Search | Text View

United States Patent	12386803
Kind Code	B2
Date of Patent	August 12, 2025
Inventor(s)	Jain; Akash et al.

Creating data in a data store using a dynamic ontology

Abstract

A method comprises storing an ontology for a data store, wherein the ontology comprises a plurality of data object types and a plurality of object property types. The method also comprises storing a validator for a particular object property type of the plurality of object property types, wherein the validator specifies a permitted value for the particular object property type, and storing a parser definition corresponding to the particular object property type, wherein the parser definition specifying a permitted format for the particular object property type. In addition, the method comprises receiving user input that changes a particular data object type of the plurality of data object types, including changing the particular object property type. Finally, the method comprises updating the validator for the particular object property type, and updating the parser definition corresponding to the particular object property type.

Inventors:	**Jain; Akash (Mountain View, CA), McGrew; Robert J. (Los Altos, CA), Gettings; Nathan (Palo Alto, CA)**
Applicant:	**Palantir Technologies Inc.** (Denver, CO)
Family ID:	**39430552**
Assignee:	**Palantir Technologies Inc. (Denver, CO)**
Appl. No.:	**18/336876**
Filed:	**June 16, 2023**

Prior Publication Data

Document Identifier	Publication Date
US 20230334032 A1	Oct. 19, 2023

Related U.S. Application Data

continuation parent-doc US 17123019 20201215 US 11714792 child-doc US 18336876
continuation parent-doc US 15448491 20170302 US 10872067 20201222 child-doc US 17123019

continuation parent-doc US 14954680 20151130 US 9589014 20170307 child-doc US 15448491
continuation parent-doc US 14508696 20141007 US 9201920 20151201 child-doc US 14954680
continuation parent-doc US 13916447 20130612 US 8856153 20141007 child-doc US 14508696
continuation parent-doc US 13106636 20110512 US 8489623 20130716 child-doc US 13916447
continuation parent-doc US 11602626 20061120 US 7962495 20110604 child-doc US 13106636

Publication Classification

Int. Cl.: **G06F16/22** (20190101); **G06F16/21** (20190101); **G06F16/23** (20190101); **G06F16/245** (20190101); **G06F16/36** (20190101); **G06F40/205** (20200101); **G06F40/211** (20200101); **G06F40/226** (20200101); **G06F40/284** (20200101)

U.S. Cl.:

CPC **G06F16/2219** (20190101); **G06F16/21** (20190101); **G06F16/213** (20190101); **G06F16/22** (20190101); **G06F16/2365** (20190101); **G06F16/245** (20190101); **G06F16/367** (20190101); **G06F40/205** (20200101); **G06F40/211** (20200101); **G06F40/226** (20200101); **G06F40/284** (20200101);

Field of Classification Search

CPC: G06F (16/2219); G06F (16/21); G06F (16/213); G06F (16/22); G06F (16/2365); G06F (16/245); G06F (16/367); G06F (40/205); G06F (40/211); G06F (40/226); G06F (40/284)

USPC: 707/755

References Cited

U.S. PATENT DOCUMENTS

Patent No.	Issued Date	Patentee Name	U.S. Cl.	CPC
5454106	12/1994	Burns	707/999.005	G06F 16/2423
6236994	12/2000	Swartz et al.	N/A	N/A
6289338	12/2000	Stoffel et al.	N/A	N/A
6640231	12/2002	Andersen et al.	N/A	N/A
6792435	12/2003	Ruddy	707/999.102	G06F 16/284
7027974	12/2005	Busch et al.	N/A	N/A
7496569	12/2008	Theobald	707/999.009	G06F 16/24528
7519589	12/2008	Charnock et al.	N/A	N/A
7840293	12/2009	Weinrich et al.	N/A	N/A
7877421	12/2010	Berger et al.	N/A	N/A
7895315	12/2010	Weinrich et al.	N/A	N/A
8117300	12/2011	Weinrich et al.	N/A	N/A
8321045	12/2011	Weinrich et al.	N/A	N/A
10025588	12/2017	Tran	N/A	G06F 16/245
11714792	12/2022	Jain et al.	N/A	N/A
2003/0172053	12/2002	Fairweather	N/A	G06F 8/427
2003/0177112	12/2002	Gardner	N/A	N/A
2004/0126840	12/2003	Cheng et al.	N/A	N/A
2004/0158455	12/2003	Spivack et al.	N/A	N/A

2006/0036592	12/2005	Das	N/A	G06F 16/2445
2006/0036633	12/2005	Chong	N/A	G06F 16/367
2006/0053170	12/2005	Hill et al.	N/A	N/A
2006/0212474	12/2005	Mccormack et al.	N/A	N/A
2006/0259514	12/2005	Weinrich et al.	N/A	N/A
2006/0259515	12/2005	Weinrich et al.	N/A	N/A
2006/0259911	12/2005	Weinrich et al.	N/A	N/A
2006/0259912	12/2005	Weinrich	719/315	G06Q 50/04
2006/0271884	12/2005	Hurst	N/A	N/A

Primary Examiner: Brown; Sheree N

Attorney, Agent or Firm: Duane Morris LLP

Background/Summary

BENEFIT CLAIM (1) The present application claims the benefit under 35 U.S.C. § 120 as a continuation of application Ser. No. 17/123,019, filed Dec. 15, 2020, which is a continuation of application Ser. No. 15/448,491, filed Mar. 2, 2017, now U.S. Pat. No. 10,872,067, which is a continuation of application Ser. No. 14/954,680, filed Nov. 30, 2015, now U.S. Pat. No. 9,589,014, which is a continuation of application Ser. No. 14/508,696, filed Oct. 7, 2014, now U.S. Pat. No. 9,201,920, which is a continuation of application Ser. No. 13/916,447, filed Jun. 12, 2013, now U.S. Pat. No. 8,856,153, which is a continuation of application Ser. No. 13/106,636, filed May 12, 2011, now U.S. Pat. No. 8,489,623, which is a continuation of application Ser. No. 11/602,626, filed Nov. 20, 2006, now U.S. Pat. No. 7,962,495, the entire contents of which are hereby incorporated by reference for all purposes as if fully set forth herein. The applicant hereby rescinds any disclaimer of claim scope in the parent applications or the prosecution history thereof and advise the USPTO that the claims in this application may be broader than any claim in the parent applications.

FIELD OF THE INVENTION
(1) The present disclosure generally relates to techniques for creating data in a data store.
BACKGROUND
(2) 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.
(3) Computer-based database systems, such as relational database management systems, typically organize data according to a fixed structure of tables and relationships. The structure may be described using an ontology, embodied in a database schema, comprising a data model that is used to represent the structure and reason about objects in the structure.
(4) An ontology of a database is normally fixed at the time that the database is created. Any change in the ontology represented by the schema is typically extremely disruptive to the database system and requires a database administrator to modify tables or relationships, or create new tables or relationships.
(5) The rigidity of the typical database ontology is a serious drawback for organizations that require flexible and dynamic data processing techniques according to changes in the data that is collected. For example, intelligence analysis is poorly suited to conventional fixed ontology systems.
SUMMARY

(6) In one embodiment, a method comprises creating and storing an ontology for a data store in response to receiving first user input defining the ontology, wherein the ontology comprises a plurality of data object types and a plurality of object property types; creating one or more parser definitions in response to receiving second user input defining the parser definitions, wherein each of the parser definitions specifies one or more sub-definitions of how to transform first input data into modified input data that is compatible with one of the object property types; storing each of the one or more parser definitions in association with one of the plurality of object property types; wherein the machine-executed operation is at least one of (a) sending said instructions over transmission media, (b) receiving said instructions over transmission media, (c) storing said instructions onto a machine-readable storage medium, and (d) executing the instructions.
(7) In one feature, the method further comprises receiving the first input data; determining whether the first input data matches one of the parser sub-definitions; using a matching one of the parser sub-definitions, creating and storing the modified input data; storing the modified input data in a property of the property type that is identified in the matching one of the parser sub-definitions.
(8) In another feature, creating and storing one or more parser definitions comprises creating and storing one or more program code modules, wherein each of the code modules comprises computer program code which when executed causes transforming the first input data into the modified input data.
(9) In another feature, creating and storing one or more parser definitions comprises creating and storing one or more transformation expressions, wherein each of the transformation expressions comprises one or more syntactic patterns and a property type identifier associated with each of the syntactic patterns.
(10) In yet another feature, creating and storing one or more parser definitions comprises creating and storing one or more transformation expressions, wherein each of the transformation expressions comprises one or more syntactic patterns and a property type identifier associated with each of the syntactic patterns, and the method further comprises receiving the first input data; determining whether the first input data matches one of the syntactic patterns; using a matching one of the syntactic patterns, creating and storing modified input data; storing the modified input data in a property of the property type that is identified by the property type identifier associated with the matching one of the syntactic patterns.
(11) In still another feature, creating one or more parser definitions comprises creating one or more parser definitions comprising a constraint on what modified input data is acceptable for creation of a property of one of the object property types. In a further feature, creating one or more parser definitions comprises creating one or more parser definitions comprising a default value to substitute for one component of the modified input data.
(12) In another feature, the method further comprises receiving the first input data; determining whether the first input data matches successive different ones of the parser sub-definitions until a matching parser sub-definition is identified; using a matching one of the parser sub-definitions, creating and storing the modified input data; storing the modified input data in a property of the property type that is identified in the matching one of the parser sub-definitions.
(13) According to another embodiment, a data storage system comprises a data store; an ontology coupled to the data store and comprising a plurality of data object types and a plurality of object property types; a parser coupled to the ontology and configured to receive input data and transform the input data into modified data to store in a property of one of the property types according to one or more parser definitions; wherein each of the object property types comprises one or more of the parser definitions, wherein each of the parser definitions specifies one or more sub-definitions of how to transform first input data into modified input data that is compatible with one of the object property types.
(14) According to another embodiment, an apparatus comprises means for creating and storing an ontology for a data store in response to receiving first user input defining the ontology, wherein the

ontology comprises a plurality of data object types and a plurality of object property types; means for creating one or more parser definitions in response to receiving second user input defining the parser definitions, wherein each of the parser definitions specifies one or more sub-definitions of how to transform first input data into modified input data that is compatible with one of the object property types; and means for storing each of the one or more parser definitions in association with one of the plurality of object property types.
(15) In another embodiment, a graphical user interface comprises an expression pattern field configured to accept user input specifying a transformation expression pattern that specifies how to transform first input data into modified input data; one or more parser sub-definitions each comprising: a portion of the transformation expression pattern; a combo box configured to accept user input specifying one of a plurality of object property component types of an ontology of a data store; wherein each of the parser sub-definitions specifies how to transform a portion of the first input data into a portion of modified input that can be stored in the specified component of one of the plurality of object property types.
(16) In one feature, the one or more parser sub-definitions comprise a constraint on how to transform the portion of the first input data into the portion of modified input data that is compatible with one of the object property types. In yet another feature, the one or more parser sub-definitions comprise a default value to substitute for the modified input data if it is empty.

Description

BRIEF DESCRIPTION OF THE DRAWINGS
(1) The present invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings and in which like reference numerals refer to similar elements and in which:
(2) FIG. **1** illustrates a system for creating data in a data store using a dynamic ontology;
(3) FIG. **2** illustrates defining a dynamic ontology for use in creating data in a data store;
(4) FIG. **3** illustrates a method of transforming data and creating the data in a data store using a dynamic ontology;
(5) FIG. **4** illustrates an example object type editor;
(6) FIG. **5**A illustrates an example parser editor;
(7) FIG. **5**B illustrates an example property editing wizard in which multiple parsers have been created for a particular property; and
(8) FIG. **6** illustrates a computer system with which an embodiment may be implemented.
DESCRIPTION OF EXAMPLE EMBODIMENTS
(9) In the following description, for the purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent, however, that the present invention may be practiced without these specific details. In other instances, well-known structures and devices are shown in block diagram form in order to avoid unnecessarily obscuring the present invention. In an embodiment, a user of a database system specifies an ontology of the database in terms of object types and property types for properties of the objects. The user further specifies how to parse input data for the database and how to map the parsed data into database elements such as objects or object properties. The database is chosen as an example embodiment, other embodiments such as flat files or search indexes could be considered as well.
(10) I. Dynamic Ontology Database System
(11) FIG. **1** illustrates a system for creating data in a database using a dynamic ontology. A parser **102** is coupled to an ontology **106**, which is coupled to a database **108**. In an embodiment, ontology **106** comprises stored information providing a data model of data stored in database **108**, and the

ontology is defined by one or more object types **110** and one or more property types **116**. One or more objects **112** in the database **108** may be instantiated based on the object types **110**, and each of the objects has one or more properties **114**A, **114**B that are instantiated based on property types **116**. The property types **116** each may comprise one or more components **118**, such as a string, number, etc. Property types **116** may be instantiated based on a base type **120**. For example, a base type **120** may be "Locations" and a property type **116** may be "Home."
(12) In an embodiment, a user of the system uses an object type editor **124** to create the object types **110** and define attributes of the object types. In an embodiment, a user of the system uses a property type editor **126** to create the property types **116** and define attributes of the property types.
(13) In an embodiment, creating a property type **116** using the property type editor **126** involves defining at least one parser definition using a parser editor **122**. A parser definition comprises metadata that informs parser **102** how to parse input data **100** to determine whether values in the input data can be assigned to the property type **116** that is associated with the parser definition. In an embodiment, each parser definition may comprise a regular expression parser **104**A or a code module parser **104**B. In other embodiments, other kinds of parser definitions may be provided using scripts or other programmatic elements. The elements of a regular expression parser **104**A and a code module parser **104**B are described further in subsequent sections. Once defined, both a regular expression parser **104**A and a code module parser **104**B can provide input to parser **102** to control parsing of input data **100**.
(14) In one embodiment of using the system of FIG. **1**, input data **100** is provided to parser **102**. An object-property mapping for the input data **100** enables the parser to determine which object type **110** should receive data from a row of the input data, and which property types **116** should receive data from individual field values in the input data. Based on the object-property mapping **101**, the parser **102** selects one of the parser definitions that is associated with a property type in the input data. The parser parses an input data field using the selected parser definition, resulting in creating modified data **103**. The modified data **103** is added to the database **108** according to ontology **106** by storing values of the modified data in a property of the specified property type. As a result, input data **100** having varying format or syntax can be created in database **108**. The ontology **106** may be modified at any time using object type editor **124** and property type editor **126**. Parser editor **122** enables creating multiple parser definitions that can successfully parse input data **100** having varying format or syntax and determine which property types should be used to transform input data **100** into modified input data **103**.
(15) II. Defining a Dynamic Ontology
(16) FIG. **2** illustrates defining a dynamic ontology for use in creating data in a database. For purposes of illustrating a clear example, steps **202-209** of FIG. **2** are first described at a high level, and details of an example implementation follow the high level description.
(17) In step **202**, one or more object types are created for a database ontology. In step **206**, one or more property types are created for each object type. As indicated in step **204**, the attributes of object types or property types of the ontology may be edited or modified at any time.
(18) In step **208**, at least one parser definition is created for each property type. At step **209**, attributes of a parser definition may be edited or modified at any time.
(19) In an embodiment, each property type is declared to be representative of one or more object types. A property type is representative of an object type when the property type is intuitively associated with the object type. For example, a property type of "Social Security Number" may be representative of an object type "Person" but not representative of an object type "Business."
(20) In an embodiment, each property type has one or more components and a base type. In an embodiment, a property type may comprise a string, a date, a number, or a composite type consisting of two or more string, date, or number elements. Thus, property types are extensible and can represent complex data structures. Further, a parser definition can reference a component of a complex property type as a unit or token.

(21) An example of a property having multiple components is a Name property having a Last Name component and a First Name component. An example of raw input data is “Smith, Jane”. An example parser definition specifies an association of input data to object property components as follows: {LAST_NAME}, {FIRST_NAME}.fwdarw.Name: Last, Name: First. In an embodiment, the association {LAST_NAME}, {FIRST_NAME} is defined in a parser definition using regular expression symbology. The association {LAST_NAME}, {FIRST_NAME} indicates that a last name string followed by a first name string comprises valid input data for a property of type Name. In contrast, input data of “Smith Jane” would not be valid for the specified parser definition, but a user could create a second parser definition that does match input data of “Smith Jane”. The definition Name: Last, Name: First specifies that matching input data values map to components named “Last” and “First” of the Name property.
(22) As a result, parsing the input data using the parser definition results in assigning the value “Smith” to the Name: Last component of the Name property, and the value “Jane” to the Name: First component of the Name property.
(23) In an embodiment, administrative users use an administrative editor to create or edit object types and property types. In an embodiment, users use the administrative editor to specify parser definitions and to associate regular expressions, code modules or scripts with the parser definitions. In the administrative editor, a user can specify attributes and components of a property type. For example, in one embodiment a user specifies a graphical user interface icon that is associated with the property type and displayed in a user interface for selecting the property type. The user further specifies a parser definition that is associated with the property type and that can parse input data and map the input data to properties corresponding to the property type. The user further specifies a display format for the property type indicating how users will see properties of that property type.
(24) FIG. **4** illustrates an example object type editor. In an embodiment, an object type editor panel **402** comprises graphical buttons **404** for selecting add, delete, and edit functions, and one or more rows **406** that identify object types and a summary of selected attributes of the object types. Example selected attributes that can be displayed in object editor panel **402** include an object type name **408** (for example, “Business”), a uniform resource identifier (URI) **410** specifying a location of information defining the object type (for example, “com.palantir.object.business”), and a base type **412** of the object type, also expressed in URI format (for example, “com.palantir.object.entity”). Each URI also may include a graphical icon **414**.
(25) In an embodiment, a user interacts with a computer to perform the following steps to define an object type. Assume for purposes of an example that the new object type is Vehicle. Using the object type editor, the user selects the “Add Object Type” button **404** and the computer generates and displays a panel that prompts the user to enter values for a new object type. The user selects a base object type of Entity, which may comprise any person, place or thing. The user assigns a graphical icon to the Vehicle object type. The user assigns a display name of “Vehicle” to the object type.
(26) In an embodiment, a user interacts with the computer to define a property type in a similar manner. The user specifies a name for the property type, a display name, and an icon. The user may specify one or more validators for a property type. Each validator may comprise a regular expression that input data modified by a parser must match to constitute valid data for that property type. In an embodiment, each validator is applied to input data before a process can store the modified input data in an object property of the associated property type. Validators are applied after parsing and before input data is allowed to be stored in an object property.
(27) In various embodiments, validators may comprise regular expressions, a set of fixed values, or a code module. For example, a property type that is a number may have a validator comprising a regular expression that matches digits 0 to 9. As another example, a property type that is a US state may have a validator that comprises the set {AK, AL, CA . . . VA} of valid two-letter postal abbreviations for states. Validator sets may be extendible to allow a user to add further values. A

property type may have component elements, and each component element may have a different validator. For example, a property type of “Address” may comprise as components “City”, “State”, and “ZIP”, each of which may have a different validator.
(28) In an embodiment, defining a property type includes identifying one or more associated words for the property type. The associated words support search functions in large database systems. For example, a property type of “Address” may have an associated word of “home” so that a search in the system for “home” properties will yield “Address” as one result.
(29) In an embodiment, defining a property type includes identifying a display formatter for the property type. A display formatter specifies how to print or display a property type value.
(30) In an embodiment, the parser definitions each include a regular expression that matches valid input, and the parser uses a regular expression processing module. For example, conventional Java language processors typically have regular expression processing modules built in. In an embodiment, parser definitions comprising regular expressions may be chained together. In another embodiment, one or more of the parser definitions each include a code module that contains logic for parsing input data and determining whether the input data matches a specified syntax or data model. The code module may be written in Java, JavaScript, or any other suitable source language.
(31) In an embodiment, there may be any number of parser definitions and sub-definitions. The number of parser definitions is unimportant because the input data is applied successively to each parser definition until a match occurs. When a match occurs, the input data is mapped using the parser sub definitions to one or more components of an instance of an object property. As a result, input data can vary syntactically from a desired syntax but correct data values are mapped into correct object property values in a database.
(32) Accordingly, referring again to FIG. **2**, creating a parser definition for a property type at step **208** may comprise selecting a parser type such as a regular expression, code module, or other parser type. When the parser type is “code module,” then a user specifies the name of a particular code module, script, or other functional element that can perform parsing for the associated property type.
(33) In an embodiment, defining a property type includes creating a definition of a parser for the property type using a parser editor. FIG. **5**A illustrates an example parser editor user interface screen display. In an embodiment, screen display **502** comprises a Parser Type combo box **504** that can receive a user selection of a parser type, such as “Regular Expression” or “Code Module.” Screen display **502** further comprises a Name text entry box **506** that can receive a user-specified name for the parser definition.
(34) When the parser type is “regular expression,” steps **214-220** are performed. At step **214**, regular expression text is specified. For example, when the Parser Type value of combo box **504** is “Regular Expression,” screen display **502** comprises an Expression Pattern text box **508** that can receive a user entry of regular expression pattern text.
(35) In step **216**, a property type component and a matching sub-definition of regular expression text is specified. For example, screen display **502** further comprises one or more property type component mappings **510**. Each property type component mapping associates a sub-definition of the regular expression pattern text with the property type component that is shown in a combo box **512**. A user specifies a property type component by selecting a property type component using combo box **512** for an associated sub-definition **513**. As shown in step **218**, specifying a property type component and sub-definition of regular expression text may be repeated for all other property type components of a particular property type. As seen in the example of FIG. **5**A, six (6) property type component mappings **510** have been created for different property types (ADDRESS1, ADDRESS2, ADDRESS3, CITY, STATE, ZIP).
(36) In step **220**, a user may specify one or more constraints, default values, and/or other attributes of a parser definition. In the example of FIG. **5**A, the user also may specify that a match to a particular property type component is not required by checking a “Not Required” check box **514**.

Screen display **502** may further comprise a Default Value text box **514** that can receive user input for a default value for the property type component. If a Default Value is specified, then the associated property type receives that value if no match occurs for associated grouping of the regular expression. In alternative embodiments, other constraints may be specified.
(37) At step **222**, the parser definition is stored in association with a property type. For example, selecting the SAVE button **520** of FIG. **5**A causes storing a parser definition based on the values entered in screen display **502**. Parser definitions may be stored in database **108**.
(38) For purposes of illustrating a clear example, the approach of FIG. **2** has been described with reference to FIG. **5**A. However, the approach of FIG. **2** may be implemented using other mechanisms for creating and specifying the values and elements identified in FIG. **2**, and the particular GUI of FIG. **5**A is not required.
(39) III. Creating Data in a Database Using a Dynamic Ontology
(40) FIG. **3** illustrates a method of transforming data and creating the data in a database using a dynamic ontology. For purposes of illustrating a clear example, the approach of FIG. **3** is described herein with reference to FIG. **1**. However, the approach of FIG. **3** may be implemented using other mechanisms for performing the functional steps of FIG. **3**, and the particular system of FIG. **1** is not required.
(41) In step **302**, input data is received. In an embodiment, an input data file is received. The input data file may comprise a comma-separated value (CSV) file, a spreadsheet, XML or other input data file format. Input data **100** of FIG. **1** may represent such file formats or any other form of input data.
(42) In step **304**, an object type associated with input data rows of the input data is identified, and one or more property types associated with input data fields of the input data are identified. For example, the object-property mapping **101** of FIG. **1** specifies that input data **100** comprises rows corresponding to object type PERSON and fields corresponding to property type components LAST_NAME, FIRST_NAME of property type NAME. The object-property mapping **101** may be integrated into input data **100** or may be stored as metadata in association with a data input tool.
(43) In step **306**, a row of data is read from the input data, and one or more field values are identified based on delimiters or other field identifiers in the input data.
(44) In step **308**, a set of parser definitions associated with the property type of a particular input data field is selected. For example, metadata stored as part of creating a property type specifies a set of parser definitions, as previously described in connection with FIG. **5**A.
(45) In step **310**, the next parser definition is applied to an input data field value. Thus, data fields are read from each row of the file and matched to each parser that has been defined for the corresponding property types. For example, assume that the mapping indicates that an input data CSV file comprises (Last Name, First Name) values for Name properties of Person objects. Data fields are read from the input data CSV file and compared to each of the parsers that has been defined for the Name property type given the First Name field and Last Name field. If a match occurs for a (Last Name, First Name) pair value to any of the parsers for the Name property type, then the parser transforms the input data pair of (,Last Name, First Name) into modified input data to be stored in an instantiation of a Name property.
(46) If applying a definition at step **310** results in a match to the input data, as tested at step **312**, then at step **318** a property instance is created, and the input data field value is stored in a property of the property type associated with the matching sub-definition of the parser definition. For example, referring to FIG. **5**A, assume that the input data matches the regular expression **508** for an ADDRESS value. The mapping **510** specifies how to store the data matching each grouping of the regular expression into a component of the ADDRESS property. In response, an instance of an ADDRESS property is created in computer memory and the matching modified input data value is stored in each component of the property instance.
(47) If no match occurs at step **312**, then control transfers to step **314** to test whether other parser

definitions match the same input data value. FIG. **5**B illustrates an example property editing wizard in which multiple parsers have been created for a particular property, and through the loop shown in FIG. **3**, each of the multiple parsers can be used in matching input data. If no match occurs to the given parser definition, then any other parser definitions for that property type are matched until either no match occurs, or no other parser definitions are available.
(48) If a grouping is empty, then the component is filled by the default value for that component, if it exists. If no other parser definitions are available, then control transfers from step **314** to step **316**, at which point an error is raised or the property is discarded
(49) At step **320**, the preceding steps are repeated for all other values and rows in the input data until the process has transformed all the input data into properties in memory.
(50) At step **322**, an object of the correct object type is instantiated. For example, the object-property mapping **101** may specify an object type for particular input data, and that type of object is instantiated. The newly created object is associated in memory with the properties that are already in memory. The resulting object is stored in the database in step **324**.
(51) Steps in the preceding process may be organized in a pipeline. Using the approaches herein, a user can self-define a database ontology and use automated, machine-based techniques to transform input data according to user-defined parsers and store the transformed data in the database according to the ontology. The approach provides efficient movement of data into a database according to an ontology. The input data has improved intelligibility after transformation because the data is stored in a canonical ontology. Further, the approach is flexible and adaptable, because the user can modify the ontology at any time and is not tied to a fixed ontology. The user also can define multiple parsers to result in semantic matches to input data even when the syntax of the input data is variable.
(52) IV. Example Implementation Hardware
(53) FIG. **6** is a block diagram that illustrates a computer system **600** upon which an embodiment of the invention may be implemented. Computer system **600** includes a bus **602** or other communication mechanism for communicating information, and a processor **604** coupled with bus **602** for processing information. Computer system **600** also includes a main memory **606**, such as a random access memory (RAM) or other dynamic storage device, coupled to bus **602** for storing information and instructions to be executed by processor **604**. Main memory **606** also may be used for storing temporary variables or other intermediate information during execution of instructions to be executed by processor **604**. Computer system **600** further includes a read only memory (ROM) **608** or other static storage device coupled to bus **602** for storing static information and instructions for processor **604**. A storage device **610**, such as a magnetic disk or optical disk, is provided and coupled to bus **602** for storing information and instructions.
(54) Computer system **600** may be coupled via bus **602** to a display **612**, such as a cathode ray tube (CRT), for displaying information to a computer user. An input device **614**, including alphanumeric and other keys, is coupled to bus **602** for communicating information and command selections to processor **604**. Another type of user input device is cursor control **616**, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor **604** and for controlling cursor movement on display **612**. 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.
(55) The invention is related to the use of computer system **600** for implementing the techniques described herein. According to one embodiment of the invention, those techniques are performed by computer system **600** in response to processor **604** executing one or more sequences of one or more instructions contained in main memory **606**. Such instructions may be read into main memory **606** from another machine-readable medium, such as storage device **610**. Execution of the sequences of instructions contained in main memory **606** causes processor **604** 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 to implement the invention. Thus, embodiments of the invention are not limited to any specific combination of hardware circuitry and software.
(56) The term “machine-readable medium” as used herein refers to any medium that participates in providing data that causes a machine to operation in a specific fashion. In an embodiment implemented using computer system **600**, various machine-readable media are involved, for example, in providing instructions to processor **604** for execution. Such a medium may take many forms, including but not limited to, non-volatile media, volatile media, and transmission media. Non-volatile media includes, for example, optical or magnetic disks, such as storage device **610**. Volatile media includes dynamic memory, such as main memory **606**. Transmission media includes coaxial cables, copper wire and fiber optics, including the wires that comprise bus **602**. Transmission media can also take the form of acoustic or light waves, such as those generated during radio wave and infrared data communications. All such media must be tangible to enable the instructions carried by the media to be detected by a physical mechanism that reads the instructions into a machine.
(57) Common forms of machine-readable media include, for example, a floppy disk, a flexible disk, hard disk, magnetic tape, or any other magnetic medium, a CD-ROM, any other optical medium, punch cards, paper tape, any other physical medium with patterns of holes, a RAM, a PROM, and EPROM, a FLASH-EPROM, any other memory chip or cartridge, a carrier wave as described hereinafter, or any other medium from which a computer can read.
(58) Various forms of machine-readable media may be involved in carrying one or more sequences of one or more instructions to processor **604** for execution. For example, the instructions may initially be carried on a magnetic disk 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 **600** can receive the data on the telephone line and use an infrared transmitter to convert the data to an infrared signal. An infrared detector can receive the data carried in the infrared signal and appropriate circuitry can place the data on bus **602**. Bus **602** carries the data to main memory **606**, from which processor **604** retrieves and executes the instructions. The instructions received by main memory **606** may optionally be stored on storage device **610** either before or after execution by processor **604**.
(59) Computer system **600** also includes a communication interface **618** coupled to bus **602**. Communication interface **618** provides a two-way data communication coupling to a network link **620** that is connected to a local network **622**. For example, communication interface **618** may be an integrated services digital network (ISDN) card or a modem to provide a data communication connection to a corresponding type of telephone line. As another example, communication interface **618** may be a local area network (LAN) card to provide a data communication connection to a compatible LAN. Wireless links may also be implemented. In any such implementation, communication interface **618** sends and receives electrical, electromagnetic or optical signals that carry digital data streams representing various types of information.
(60) Network link **620** typically provides data communication through one or more networks to other data devices. For example, network link **620** may provide a connection through local network **622** to a host computer **624** or to data equipment operated by an Internet Service Provider (ISP) **626**. ISP **626** in turn provides data communication services through the worldwide packet data communication network now commonly referred to as the “Internet” **628**. Local network **622** and Internet **628** both use electrical, electromagnetic or optical signals that carry digital data streams. The signals through the various networks and the signals on network link **620** and through communication interface **618**, which carry the digital data to and from computer system **600**, are exemplary forms of carrier waves transporting the information.
(61) Computer system **600** can send messages and receive data, including program code, through the network(s), network link **620** and communication interface **618**. In the Internet example, a

server **630** might transmit a requested code for an application program through Internet **628**, ISP **626**, local network **622** and communication interface **618**.
(62) The received code may be executed by processor **604** as it is received, and/or stored in storage device **610**, or other non-volatile storage for later execution. In this manner, computer system **600** may obtain application code in the form of a carrier wave.
(63) In the foregoing specification, embodiments of the invention have been described with reference to numerous specific details that may vary from implementation to implementation. Thus, the sole and exclusive indicator of what is the invention, and is intended by the applicants to be the invention, is the set of claims that issue from this application, in the specific form in which such claims issue, including any subsequent correction. Any definitions expressly set forth herein for terms contained in such claims shall govern the meaning of such terms as used in the claims. Hence, no limitation, element, property, feature, advantage or attribute that is not expressly recited in a claim should limit the scope of such claim in any way. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.

Claims

1. A computer-implemented method of managing a dynamic ontology, comprising: storing, by a processor, an ontology for a data store, wherein the ontology comprises a plurality of data object types and a plurality of object property types; storing, by the processor, a validator for a particular object property type of the plurality of object property types, the validator specifying a permitted value for the particular object property type; storing a parser definition corresponding to the particular object property type, the parser definition specifying a permitted format for the particular object property type; receiving user input that changes a particular data object type of the plurality of data object types, including changing the particular object property type; updating the validator for the particular object property type; updating the parser definition corresponding to the particular object property type.
2. The computer-implemented method of claim 1, the user input changing a name, a location, a base type, or a graphical icon of the particular data object type.
3. The computer-implemented method of claim 1, the user input changing a set of associated words or a display formatter for the particular object property type.
4. The computer-implemented method of claim 1, updating the parser definition comprising receiving a user selection of a new parser type from a plurality of parser types, including a regular expression or a code module.
5. The computer-implemented method of claim 1, updating the parser definition comprising receiving a user specification of one or more constraints, default values, or other attributes of the parser definition.
6. The computer-implemented method of claim 1, updating the parser definition comprising receiving a user specification of a regular expression.
7. The computer-implemented method of claim 1, updating the validator comprising receiving a user specification of a regular expression.
8. The computer-implemented method of claim 1, further comprising: receiving input data; creating modified data from the input data using the parser definition; determining that the modified data matches the validator; storing, in response to the determining, the modified data in an object property instance of the particular object property type.
9. The computer-implemented method of claim 1, further comprising: receiving input data; identifying a data object type associated with a row of one or more rows of the input data and an object property type associated with each field of one or more fields of the row; selecting a set of parser definitions associated with the one or more object property types.
10. The computer-implemented method of claim 9, further comprising: for each parser definition of

the set of parser definitions, determining that the parser definition matches a corresponding field and creating a property instance of a corresponding object property type; creating an object instance of the data object type; storing the object instance in a database.
11. A computer-readable, non-transitory storage medium storing computer-executable instructions, which when executed cause one or more processors to perform a method of managing a dynamic ontology, the method comprising: storing an ontology for a data store, wherein the ontology comprises a plurality of data object types and a plurality of object property types; storing a validator for a particular object property type of the plurality of object property types, the validator specifying a permitted value for the particular object property type; storing a parser definition corresponding to the particular object property type, the parser definition specifying a permitted format for the particular object property type; receiving user input that changes a particular data object type of the plurality of data object types, including changing the particular object property type; updating the validator for the particular object property type; updating the parser definition corresponding to the particular object property type.
12. The computer-readable, non-transitory storage medium of claim 11, the user input changing a name, a location, a base type, or a graphical icon of the particular data object type.
13. The computer-readable, non-transitory storage medium of claim 11, the user input changing a set of associated words or a display formatter for the particular object property type.
14. The computer-readable, non-transitory storage medium of claim 11, updating the parser definition comprising receiving a user selection of a new parser type from a plurality of parser types, including a regular expression or a code module.
15. The computer-readable, non-transitory storage medium of claim 11, updating the parser definition comprising receiving a user specification of one or more constraints, default values, or other attributes of the parser definition.
16. The computer-readable, non-transitory storage medium of claim 11, updating the parser definition comprising receiving a user specification of a regular expression.
17. The computer-readable, non-transitory storage medium of claim 11, updating the validator comprising receiving a user specification of a regular expression.
18. The computer-readable, non-transitory storage medium of claim 11, the method further comprising: receiving input data; creating modified data from the input data using the parser definition; determining that the modified data matches the validator; storing, in response to the determining, the modified data in an object property instance of the particular object property type.
19. The computer-readable, non-transitory storage medium of claim 11, the method further comprising: receiving input data; identifying a data object type associated with a row of one or more rows of the input data and an object property type associated with each field of one or more fields of the row; selecting a set of parser definitions associated with the one or more object property types.
20. The computer-readable, non-transitory storage medium of claim 19, the method further comprising: for each parser definition of the set of parser definitions, determining that the parser definition matches a corresponding field and creating a property instance of a corresponding object property type; creating an object instance of the data object type; storing the object instance in a database.