A METHOD, COMPUTER PROGRAM AND COMPUTER FOR ACCESSING DATA IN AN ENVIRONMENT OF MULTIPLE DATA REPOSITORIES

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention relates to a method, computer program and computer for accessing data in an environment of multiple data repositories.

Description of the Related Art

Databases are computerized information storage and retrieval systems. A relational database management system is a computer database management system (DBMS) that uses relational techniques for storing and retrieving data. The most prevalent type of database is the relational database, a tabular database in which data is defined so that it can be reorganized and accessed in a number of different ways.

Regardless of the particular architecture, in a DBMS, a requesting entity (e.g., an application, the operating system or a user) demands access to a specified database by issuing a database access request. Such requests may include, for instance, simple catalog lookup requests or transactions and combinations of transactions that operate to read, change and add specified records in the database. These requests are made using high-level query languages such as the Structured Query Language (SQL). Illustratively, SQL is used to make interactive queries for getting information from and updating a database such as International Business Machines' (IBM) DB2, Microsoft's SQL Server, and database products from Oracle, Sybase, and Computer Associates. The term "query" denominates a set of commands for retrieving data from a stored database. Queries take the form of a command language that lets programmers and programs select, insert, update, find out the location of data, and so forth.

One of the issues faced by data mining and database query applications, in general, is their close relationship with a given database schema (e.g., a relational database schema). This relationship makes it difficult to support an application as changes are made to the corresponding underlying database schema. Further, the migration of the application to alternative underlying data representations is inhibited. In today's environment, the foregoing disadvantages are largely due to the

reliance applications have on SQL, which presumes that a relational model is used to represent information being queried. Furthermore, a given SQL query is dependent upon a particular relational schema since specific database tables, columns and relationships are referenced within the SQL query representation. As a result of these limitations, a number of difficulties arise.

One difficulty is that changes in the underlying relational data model require changes to the SQL foundation that the corresponding application is built upon. Therefore, an application designer must either forego changing the underlying data model to avoid application maintenance or must change the application to reflect changes in the underlying relational model. Another difficulty is that extending an application to work with multiple relational data models requires separate versions of the application to reflect the unique SQL requirements driven by each unique relational schema. Yet another difficulty is evolution of the application to work with alternate data representations because SQL is designed for use with relational systems. Extending the application to support alternative data representations, such as XML, requires rewriting the application's data management layer to use non-SQL data access methods.

A typical approach used to address the foregoing problems is software encapsulation. Software encapsulation involves using a software interface or component to encapsulate access methods to a particular underlying data representation. An example is found in the Enterprise JavaBean™ (EJB) specification that is a component of the Java™ 2 Enterprise Edition (J2EE) suite of technologies. In the case of EJB, entity beans serve to encapsulate a given set of data, exposing a set of Application Program Interfaces (APIs) that can be used to access this information. This is a highly specialized approach requiring the software to be written (in the form of new entity EJBs) whenever a new set of data is to be accessed or when a new pattern of data access is desired. The EJB model also requires a code update, application build and deployment cycle to react to reorganization of the underlying physical data model or to support alternative data representations. EJB programming also requires specialized skills, since more advanced Java programming techniques are involved. Accordingly, the EJB approach and other similar approaches are rather inflexible and costly to maintain for general-purpose query applications accessing an evolving physical data model. (Java and all Java based trademarks are trademarks of Sun Microsystems Inc. in the United States, other countries, or both).

In addition to the difficulties of accessing heterogeneous data representations, today's environment is complicated by the fact that data is often highly distributed. Pervasive infrastructures like the Internet include a host of data sources which must be made accessible to users in order to be of value. Conventional solutions dealing with localized, homogenized data are no longer viable and developing solutions to deal with distributed and heterogeneous data is problematic because such solutions must have knowledge of the location of each data source and must provide unique logic (software) to deal with each different type of data representation. As a result, typical solutions (such as the provision of data warehouses containing all of the information required by applications using the warehouse) do not easily adapt to changes in the location or representation of the data being consumed and cannot easily be redeployed to work with a different data topology. The data warehouse also presents problems when there is a need to expand the content of the warehouse with additional, publicly available information. In some cases, the external data source may be very large and subject to change. It can be very costly to maintain a local copy of such data within a given data warehouse.

Therefore, there is a need for an improved and more flexible method for accessing data which is not limited to the particular manner in which the underlying physical data is represented.

SUMMARY OF THE INVENTION

According to one aspect, the invention provides a method of accessing data in an environment of multiple data repositories, comprising: receiving from a requesting entity, an abstract query according to a query specification of the requesting entity; wherein the query specification provides a definition for a plurality of logical fields of the abstract query; and transforming the abstract query into a query consistent with a particular physical data representation of the data according to access methods which map the logical fields to physical entities of the data by defining a method for accessing each of the physical entities and a location for each of the physical entities.

The method can of course be performed using computer software.

Preferably there is provided a method, computer program and computer for remote data access and integration of distributed data sources through data scheme and query abstraction.

The present invention is preferably directed to a method, computer and computer program for accessing data independent of the particular manner in which the data is physically represented. Preferably, abstraction layers are provided to represent various distributed data sources available for use by an application and to describe a query used by the application to access and/or update information contained in these data sources. A runtime component is preferably responsible for resolving an abstract query into concrete data access requests to one or more data repositories using information contained in a data repository abstraction component (one of the abstraction layers).

One embodiment provides a method of providing access to data having a particular physical data representation. The method comprises providing, for a requesting entity, a query specification comprising a plurality of logical fields for defining an abstract query; and providing a data repository abstraction which maps the plurality of logical fields to physical entities of the data. In one embodiment, the data repository abstraction comprises, for each logical field, at least one locator which defines a location of a physical entity of the data and an access method which defines a mechanism for accessing the physical entity of the data.

One embodiment provides a computer, comprising: a processor and a memory containing at least (i) a requesting entity comprising a query specification providing a definition for an abstract query comprising a plurality of logical fields, (ii) a data repository abstraction component comprising mapping rules which map the logical fields to physical entities of data, wherein the mapping rules comprise location specifications for each of at least a portion of the logical fields of the abstract query, and wherein each of the location specifications specify a location of a data source containing a physical entity to be accessed; and (iii) a runtime component for transforming the abstract query into a query consistent with the physical entities of data according to the mapping rules.

According to another aspect, there is provided a computer, comprising: a memory containing at least (i) a query specification providing a definition for an abstract query comprising a plurality of logical fields, (ii) a data repository abstraction component comprising mapping rules which map the logical fields to physical entities of data, wherein the mapping rules comprise location specifications for each of at least a portion of the logical fields of the abstract query, and wherein each of the location specifications specify a location of a data source

containing a physical entity to be accessed; and (iii) a runtime component for transforming the abstract query into a query consistent with the physical entities of data according to the mapping rules; and a processor adapted to execute contents of the memory.

According to one embodiment, there is provided a method of providing access to data in an environment of multiple data repositories, comprising: providing, for a requesting entity, a query specification comprising a plurality of logical fields for defining an abstract query; and for each of the plurality of logical fields, providing an access method which specifies at least a method for accessing the data and a location of the data.

Preferably the method further comprises issuing the abstract query by the requesting entity according to the query specification; transforming the abstract query into a query consistent with a particular physical data representation of the data; and accessing a data repository specified by the location in the access method for the physical entity of the data for a particular logical field of the plurality of logical fields.

Preferably the query consistent with the particular physical data representation is one of a SQL query, an XML query and a procedural request.

Preferably transforming the abstract query into the query consistent with the particular physical data representation comprises partitioning the abstract query into sub-queries grouped according to access method types.

Preferably the access method types are selected from a group comprising an SQL query type, an XML query type and a procedural request type.

According to a preferred embodiment, there is provided a method of accessing data in an environment of multiple data repositories, comprising: issuing, by a requesting entity, an abstract query according to a query specification of the requesting entity; wherein the query specification provides a definition for a plurality of logical fields of the abstract query; and transforming the abstract query into a query consistent with a particular physical data representation of the data according to access methods which map the logical fields to physical

entities of the data by defining a method for accessing each of the physical entities and a location for each of the physical entities.

Preferably a data repository, specified by the location for a physical entity of the data for a particular logical field of the plurality of logical fields, can be accessed.

Preferably the abstract query comprises at least one selection criterion and a result specification.

Preferably the method comprises for a physical entity of the data for a particular logical field of the plurality of logical fields, determining whether the physical entity of the data is located in a local cache; and if not, accessing a data repository specified by the location in the access method for the physical entity of the data.

Preferably transformation of the abstract query into the query consistent with the particular physical data representation comprises partitioning the abstract query into sub-queries grouped according to access method types.

Preferably the access method types are selected from a group comprising an SQL query type, an XML query type and a procedural request type.

According to a preferred embodiment, there is provided a computer-readable medium containing a program which, when executed by a processor, performs an operation of providing access to data in an environment of multiple data repositories, the program comprising: a query specification for a requesting entity, the query specification comprising a plurality of logical fields for defining an abstract query; and an access method for each logical field each of which defines a method for accessing a physical entity of the data and a plurality of parameters to be passed to the method for accessing the physical entity, wherein at least one parameter is a location parameter specifying a location of a data source containing the physical entity.

The requesting entity may, for example, be an application.

Preferably each of the plurality of access methods define a particular physical representation and a location of the respective physical entity of the data.

Preferably the operation comprises: issuing the abstract query by the requesting entity according to the query specification; transforming the abstract query into a query consistent with the particular physical data representation; and accessing a data repository specified by the location for the physical entity of the data for a particular logical field of the plurality of logical fields.

According to a preferred embodiment, there is provided a computer-readable medium containing a program which, when executed by a processor, performs an operation of accessing data having a particular physical data representation, the operation comprising: issuing, by a requesting entity, an abstract query according to a query specification of the requesting entity; wherein the query specification provides a definition for logical fields of the abstract query; and transforming the abstract query into a query consistent with a particular physical data representation of the data according to access methods which map the logical fields to physical entities of the data by defining, for each of the physical entities, at least a method for accessing the physical entity and a location of the physical entity.

According to a preferred embodiment, there is provided a computer, comprising: a memory containing at least (i) a requesting entity comprising a query specification providing a definition for an abstract query comprising a plurality of logical fields, (ii) a data repository abstraction component comprising mapping rules which map the logical fields to physical entities of data, wherein the mapping rules comprise location specifications for each of at least a portion of the logical fields of the abstract query, and wherein each of the location specifications specify a location of a data source containing a physical entity to be accessed; and (iii) a runtime component for transforming the abstract query into a query consistent with the physical entities of data according to the mapping rules; and a processor adapted to execute contents of the memory.

Preferably a first portion of the data sources specified by the respective location specification are local and a second portion are remote.

BRIEF DESCRIPTION OF THE DRAWINGS

Preferred embodiments of the present invention will now be described, by way of example only, and with reference to the following drawings:

FIG. 1 is a computer system illustratively utilized in accordance with preferred embodiments of the invention;

FIG. 2A is an illustrative relational view of software components;

FIG. 2B is one embodiment of an abstract query and a data repository abstraction for a relational data access;

FIG. 3 is a flow chart illustrating the operation of a runtime component in accordance with one embodiment of the present invention;

FIG. 4 is a flow chart illustrating the operation of a runtime component in accordance with one embodiment of the present invention;

FIG. 5 is an illustrative relational view of software components in which multiple sources of data are accessible;

FIG. 6 shows an illustrative abstract query 602 comprising a plurality of logical fields;

FIG. 7 is an illustrative field specification of a data repository abstraction component configured with a relational access method; and

FIG. 8 is an illustrative field specification of a data repository abstraction component configured with a procedural access method.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

INTRODUCTION

The present invention generally is directed to a system, method and article of manufacture for accessing data independent of the particular manner in which the data is physically represented. The data may comprise a plurality of different data sources.

In one embodiment, a data repository abstraction layer provides a logical view of one or more underlying data repositories that is independent of the particular manner of data representation. Where multiple data sources are provided, an instance of the data repository abstraction layer is configured with a location specification identifying the location of the data to be accessed. A query abstraction layer is also provided and is based on the data repository abstraction layer. A runtime component performs translation of an abstract query (constructed according to the query abstraction layer) into a form that can be used against a particular physical data representation.

One embodiment of the invention is implemented as a program product for use with a computer system such as, for example, the computer system 100 shown in FIG. 1 and described below. The program(s) of the program product defines functions of the embodiments (including the methods described herein) and can be contained on a variety of signal-bearing media. Illustrative signal-bearing media include, but are not limited to: (i) information permanently stored on non-writable storage media (*e.g.*, read-only memory devices within a computer such as CD-ROM disks readable by a CD-ROM drive); (ii) alterable information stored on writable storage media (*e.g.*, floppy disks within a diskette drive or hard-disk drive); or (iii) information conveyed to a computer by a communications medium, such as through a computer or telephone network, including wireless communications. The latter embodiment includes information downloaded from the Internet and other networks.

In general, the routines executed to implement the embodiments of the invention, may be part of an operating system or a specific application, component, program, module, object, or sequence of instructions. The software of the preferred embodiment typically is comprised of a multitude of instructions that will be translated by the native computer into a machine-readable format and hence executable instructions. Also, programs are comprised of variables and data structures that either reside locally to the program or are found in memory or on storage devices. However, it should be appreciated that any particular nomenclature that follows is used merely for convenience, and thus the invention should not be limited to use solely in any specific application identified and/or implied by such nomenclature.

PHYSICAL VIEW OF ENVIRONMENT

FIG. 1 depicts a block diagram of a networked system 100 in which embodiments of the present invention may be implemented. In general, the networked system 100 includes a client (e.g., user's) computer 102 (three such client computers 102 are shown) and at least one server 104 (one such server 104). The client computer 102 and the server computer 104 are connected via a network 126. In general, the network 126 may be a local area network (LAN) and/or a wide area network (WAN). In a particular embodiment, the network 126 is the Internet.

The client computer 102 includes a Central Processing Unit (CPU) 110 connected via a bus 126 to a memory 112, storage 114, an input device 116, an output device 119, and a network interface device 118. The input device 116 can be any device to give input to the client computer 102. For example, a keyboard, keypad, light-pen, touch-screen, track-ball, or speech recognition unit, audio/video player, and the like could be used. The output device 119 can be any device to give output to the user, e.g., any conventional display screen. Although shown separately from the input device 116, the output device 119 and input device 116 could be combined. For example, a display screen with an integrated touch-screen, a display with an integrated keyboard, or a speech recognition unit combined with a text speech converter could be used.

The network interface device 118 may be any entry/exit device configured to allow network communications between the client computer 102 and the server computer 104 via the network 126. For example, the network interface device 118 may be a network adapter or other network interface card (NIC).

Storage 114 is preferably a Direct Access Storage Device (DASD). Although it is shown as a single unit, it could be a combination of fixed and/or removable storage devices, such as fixed disc drives, floppy disc drives, tape drives, removable memory cards, or optical storage. The memory 112 and storage 114 could be part of one virtual address space spanning multiple primary and secondary storage devices.

The memory 112 is preferably a random access memory sufficiently large to hold programming and data structures of a preferred embodiment of the invention. While the memory 112 is shown as a single entity, it should be understood that the memory 112 may in fact comprise a plurality

of modules, and that the memory 112 may exist at multiple levels, from high speed registers and caches to lower speed but larger DRAM chips.

Illustratively, the memory 112 contains an operating system 124. Illustrative operating systems, which may be used to advantage, include Linux and Microsoft's Windows®. More generally, any operating system supporting the functions disclosed herein may be used.

The memory 112 is also shown containing a browser program 122 that, when executed on CPU 110, provides support for navigating between the various servers 104 and locating network addresses at one or more of the servers 104. In one embodiment, the browser program 122 includes a web-based Graphical User Interface (GUI), which allows the user to display Hyper Text Markup Language (HTML) information. More generally, however, the browser program 122 may be any GUI-based program capable of rendering the information transmitted from the server computer 104.

The server computer 104 may be physically arranged in a manner similar to the client computer 102. Accordingly, the server computer 104 is shown generally comprising a CPU 130, a memory 132, and a storage device 134, coupled to one another by a bus 136. Memory 132 may be a random access memory sufficiently large to hold the programming and data structures, of a preferred embodiment of the present invention, that are located on the server computer 104.

The server computer 104 is generally under the control of an operating system 138 shown residing in memory 132. Examples of the operating system 138 include IBM® OS/400®, UNIX®, Microsoft® Windows®, and the like. More generally, any operating system capable of supporting the functions described herein may be used. (IBM and OS/400 are trademarks of International Business Machines in the United States, other countries, or both; UNIX is a registered trademark of the Open Group in the United States and other countries; Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both).

The memory 132 further includes one or more applications 140 and an abstract query interface 146. The applications 140 and the abstract query interface 146 are software products comprising a plurality of instructions that are resident at various times in various memory and storage devices in the computer system 100. When read and executed by one or more processors 130 in the server 104, the applications 140 and the abstract query interface 146 cause the computer system 100 to perform the steps

necessary to execute steps or elements embodying the various aspects of a preferred embodiment of the invention. The applications 140 (and more generally, any requesting entity, including the operating system 138 and, at the highest level, users) issue queries against a database. Queries may, for example, be issued include local databases 156_1...156_N, and remote databases 157_1...157_N, collectively referred to as database(s) 156-157). Illustratively, the databases 156 are shown as part of a database management system (DBMS) 154 in storage 134. More generally, as used herein, the term "databases" refers to any collection of data regardless of the particular physical representation. By way of illustration, the databases 156-157 may be organized according to a relational schema (accessible by SQL queries) or according to an XML schema (accessible by XML queries). However, the invention is not limited to a particular schema and contemplates extension to schemas presently unknown. As used herein, the term "schema" generically refers to a particular arrangement of data.

In one embodiment, the queries issued by the applications 140 are defined according to an application query specification 142 included with each application 140. The queries issued by the applications 140 may be predefined (i.e., hard coded as part of the applications 140) or may be generated in response to input (e.g., user input). In either case, the queries (referred to herein as "abstract queries") are composed using logical fields defined by the abstract query interface 146. In particular, the logical fields used in the abstract queries are defined by a data repository abstraction component 148 of the abstract query interface 146. The abstract queries are executed by a runtime component 150 which transforms the abstract queries into a form consistent with the physical representation of the data contained in one or more of the databases 156-157. The application query specification 142 and the abstract query interface 146 are further described with reference to FIGS. 2A-B.

In one embodiment, elements of a query are specified by a user through a graphical user interface (GUI). The content of the GUIs is generated by the application(s) 140. In a particular embodiment, the GUI content is hypertext markup language (HTML) content which may be rendered on the client computer systems 102 with the browser program 122. Accordingly, the memory 132 includes a Hypertext Transfer Protocol (http) server process 138 (e.g., a web server) adapted to service requests from the client computer 102. For example, the process 138 may respond to requests to access a database(s) 156, which illustratively resides on the

server 104. Incoming client requests for data from a database 156-157 invoke an application 140. When executed by the processor 130, the application 140 causes the server computer 104 to perform the steps or elements embodying the various aspects of a preferred embodiment of the present invention, including accessing the database(s) 156-157. In one embodiment, the application 140 comprises a plurality of servlets configured to build GUI elements, which are then rendered by the browser program 122. Where the remote databases 157 are accessed via the application 140, the data repository abstraction component 148 is configured with a location specification identifying the database containing the data to be retrieved. This latter embodiment will be described in more detail below.

FIG. 1 is merely one hardware/software configuration for the networked client computer 102 and server computer 104. Embodiments of the present invention can apply to any comparable hardware configuration, regardless of whether the computer systems are complicated, multi-user computing apparatus, single-user workstations, or network appliances that do not have non-volatile storage of their own. Further, it is understood that while reference is made to particular markup languages, including HTML, the invention is not limited to a particular language, standard or version. Accordingly, persons skilled in the art will recognize that the invention is adaptable to other markup languages as well as non-markup languages and that the invention is also adaptable future changes in a particular markup language as well as to other languages presently unknown. Likewise, the http server process 138 shown in FIG. 1 is merely illustrative and other embodiments adapted to support any known and unknown protocols are contemplated.

LOGICAL/RUNTIME VIEW OF ENVIRONMENT

FIGS. 2A-B show a plurality of interrelated components of the invention in accordance with a preferred embodiment. The requesting entity (e.g., one of the applications 140) issues a query 202 as defined by the respective application query specification 142 of the requesting entity. The resulting query 202 is generally referred to herein as an "abstract query" because the query is composed according to abstract (i.e., logical) fields rather than by direct reference to the underlying physical data entities in the databases 156-157. As a result, abstract queries may be defined that are independent of the particular underlying data representation used. In one embodiment, the application query specification 142 includes both criteria used for data selection

(selection criteria 204) and an explicit specification of the fields to be returned (return data specification 206) based on the selection criteria 204.

The logical fields specified by the application query specification 142 and used to compose the abstract query 202 are defined by the data repository abstraction component 148. In general, the data repository abstraction component 148 exposes information as a set of logical fields that may be used within a query (e.g., the abstract query 202) issued by the application 140 to specify criteria for data selection and specify the form of result data returned from a query operation. The logical fields are defined independently of the underlying data representation being used in the databases 156-157, thereby allowing queries to be formed that are loosely coupled to the underlying data representation.

In general, the data repository abstraction component 148 comprises a plurality of field specifications 208_1, 208_2, 208_3, 208_4 and 208_5 (five shown by way of example), collectively referred to as the field specifications 208. Specifically, a field specification is provided for each logical field available for composition of an abstract query. Each field specification comprises a logical field name 210_1, 210_2, 210_3, 210_4, 210_5 (collectively, field name 210) and an associated access method 212_1, 212_2, 212_3, $2f12_4$, 212_5 (collectively, access method 212). The access methods associate (i.e., map) the logical field names to a particular physical data representation 214_1, 214_2...214_N in a database (e.g., one of the databases 156). By way of illustration, two data representations are shown, an XML data representation 214_1 and a relational data representation 214_2. However, the physical data representation 214_N indicates that any other data representation, known or unknown, is contemplated.

Any number of access methods are contemplated depending upon the number of different types of logical fields to be supported. In one embodiment, access methods for simple fields, filtered fields and composed fields are provided. The field specifications 208_1, 208_2 and 208_5 exemplify simple field access methods 212_1, 212_2, and 212_5, respectively. Simple fields are mapped directly to a particular entity in the underlying physical data representation (e.g., a field mapped to a given database table and column). By way of illustration, the simple field access method 212_1 shown in FIG. 2B maps the logical field name 210_1 ("FirstName") to a column named "f_name" in a table named "contact". The field specification 208_3 exemplifies a filtered field access method 212_3. Filtered fields identify an associated physical entity and provide rules used to define a

particular subset of items within the physical data representation. An example is provided in FIG. 2B in which the filtered field access method 212_3 maps the logical field name 210_3 ("AnytownLastName") to a physical entity in a column named "l_name" in a table named "contact" and defines a filter for individuals in the city of Anytown. Another example of a filtered field is a New York ZIP code field that maps to the physical representation of ZIP codes and restricts the data only to those ZIP codes defined for the state of New York. The field specification 208_4 exemplifies a composed field access method 212_4. Composed access methods compute a logical field from one or more physical fields using an expression supplied as part of the access method definition. In this way, information which does not exist in the underlying data representation may computed. In the example illustrated in FIG. 2B the composed field access method 212_3 maps the logical field name 210_3 "AgeInDecades" to "AgeInYears/10". Another example is a sales tax field that is composed by multiplying a sales price field by a sales tax rate.

It is contemplated that the formats for any given data type (e.g., dates, decimal numbers, etc.) of the underlying data may vary. Accordingly, in one embodiment, the field specifications 208 include a type attribute which reflects the format of the underlying data. However, in another embodiment, the data format of the field specifications 208 is different from the associated underlying physical data, in which case an access method is responsible for returning data in the proper format assumed by the requesting entity. Thus, the access method should preferably know what format of data is assumed (i.e., according to the logical field) as well as the actual format of the underlying physical data. The access method can then convert the underlying physical data into the format of the logical field.

By way of example, the field specifications 208 of the data repository abstraction component 148 shown in FIG. 2 are representative of logical fields mapped to data represented in the relational data representation 214_2. However, other instances of the data repository abstraction component 148 map logical fields to other physical data representations, such as XML. Further, in one embodiment, a data repository abstraction component 148 is configured with access methods for procedural data representations. One embodiment of such a data repository abstraction component 148 is described below with respect to FIG. 8.

An illustrative abstract query corresponding to the abstract query 202 shown in FIG. 2 is shown in Table I below. By way of illustration,

the data repository abstraction 148 is defined using XML. However, any other language may be used to advantage

TABLE I - QUERY EXAMPLE

```
001  <?xml version="1.0"?>
002  <!--Query string representation: (FirstName = "Mary" AND LastName =
003  "McGoon") OR State = "NC"-->
004  <QueryAbstraction>
005   <Selection>
006    <Condition internalID="4">
007     <Condition field="FirstName" operator="EQ" value="Mary"
008  internalID="1"/>
009     <Condition field="LastName" operator="EQ" value="McGoon"
010  internalID="3" relOperator="AND"></Condition>
011    </Condition>
012    <Condition field="State" operator="EQ" value="NC" internalID="2"
013  relOperator="OR"></Condition>
014   </Selection>
015   <Results>
016     <Field name="FirstName"/>
017     <Field name="LastName"/>
018     <Field name="State"/>
019   </Results>
020  </QueryAbstraction>
```

Illustratively, the abstract query shown in Table I includes a selection specification (lines 005-014) containing selection criteria and a results specification (lines 015-019). In one embodiment, a selection criterion consists of a field name (for a logical field), a comparison operator (=, >, <, etc) and a value expression (what is the field being compared to). In one embodiment, result specification is a list of abstract fields that are to be returned as a result of query execution. A result specification in the abstract query may consist of a field name and sort criteria.

An illustrative instance of a data repository abstraction component 148 corresponding to the abstract query in Table I is shown in Table II below. By way of illustration, the data repository abstraction component 148 is defined using XML. However, any other language may be used to advantage.

TABLE II - DATA REPOSITORY ABSTRACTION EXAMPLE

```
001  <?xml version="1.0"?>
002  <DataRepository>
003   <Category name="Demographic">
004    <Field queryable="Yes" name="FirstName" displayable="Yes">
005      <AccessMethod>
006        <Simple columnName="f_name" tableName="contact"></Simple>
007      </AccessMethod>
```

```
008             <Type baseType="char"></Type>
009         </Field>
010         <Field queryable="Yes" name="LastName" displayable="Yes">
011             <AccessMethod>
012               <Simple columnName="l_name" tableName="contact"></Simple>
013             </AccessMethod>
014             <Type baseType="char"></Type>
015         </Field>
016         <Field queryable="Yes" name="State" displayable="Yes">
017             <AccessMethod>
018               <Simple columnName="state" tableName="contact"></Simple>
019             </AccessMethod>
020             <Type baseType="char"></Type>
021         </Field>
022       </Category>
023     </DataRepository>
```

FIG. 3 shows an illustrative runtime method 300 exemplifying one embodiment of the operation of the runtime component 150. The method 300 is entered at step 302 when the runtime component 150 receives as input an instance of an abstract query (such as the abstract query 202 shown in FIG. 2). At step 304, the runtime component 150 reads and parses the instance of the abstract query and locates individual selection criteria and desired result fields. At step 306, the runtime component 150 enters a loop (comprising steps 306, 308, 310 and 312) for processing each query selection criteria statement present in the abstract query, thereby building a data selection portion of a Concrete Query. In one embodiment, a selection criterion consists of a field name (for a logical field), a comparison operator (=, >, <, etc) and a value expression (what is the field being compared to). At step 308, the runtime component 150 uses the field name from a selection criterion of the abstract query to look up the definition of the field in the data repository abstraction 148. As noted above, the field definition includes a definition of the access method used to access the physical data associated with the field. The runtime component 150 then builds (step 310) a Concrete Query Contribution for the logical field being processed. As defined herein, a Concrete Query Contribution is a portion of a concrete query that is used to perform data selection based on the current logical field. A concrete query is a query represented in languages like SQL and XML Query and is consistent with the data of a given physical data repository (e.g., a relational database or XML repository). Accordingly, the concrete query is used to locate and retrieve data from a physical data repository, represented by the databases 156-157 shown in FIG. 1. The Concrete Query Contribution generated for the current field is then added to a Concrete Query Statement. The method 300 then returns to step 306 to begin processing for the next field of the abstract query. Accordingly, the process

entered at step 306 is iterated for each data selection field in the abstract query, thereby contributing additional content to the eventual query to be performed.

After building the data selection portion of the concrete query, the runtime component 150 identifies the information to be returned as a result of query execution. As described above, in one embodiment, the abstract query defines a list of abstract fields that are to be returned as a result of query execution, referred to herein as a result specification. A result specification in the abstract query may consist of a field name and sort criteria. Accordingly, the method 300 enters a loop at step 314 (defined by steps 314, 316, 318 and 320) to add result field definitions to the concrete query being generated. At step 316, the runtime component 150 looks up a result field name (from the result specification of the abstract query) in the data repository abstraction 148 and then retrieves a Result Field Definition from the data repository abstraction 148 to identify the physical location of data to be returned for the current logical result field. The runtime component 150 then builds (as step 318) a Concrete Query Contribution (of the concrete query that identifies physical location of data to be returned) for the logical result field. At step 320, Concrete Query Contribution is then added to the Concrete Query Statement. Once each of the result specifications in the abstract query has been processed, the query is executed at step 322.

One embodiment of a method 400 for building a Concrete Query Contribution for a logical field according to steps 310 and 318 is described with reference to FIG. 4. At step 402, the method 400 queries whether the access method associated with the current logical field is a simple access method. If so, the Concrete Query Contribution is built (step 404) based on physical data location information and processing then continues according to method 300 described above. Otherwise, processing continues to step 406 to query whether the access method associated with the current logical field is a filtered access method. If so, the Concrete Query Contribution is built (step 408) based on physical data location information for some physical data entity. At step 410, the Concrete Query Contribution is extended with additional logic (filter selection) used to subset data associated with the physical data entity. Processing then continues according to method 300 described above.

If the access method is not a filtered access method, processing proceeds from step 406 to step 412 where the method 400 queries whether the access method is a composed access method. If the access method is a

composed access method, the physical data location for each sub-field reference in the composed field expression is located and retrieved at step 414. At step 416, the physical field location information of the composed field expression is substituted for the logical field references of the composed field expression, whereby the Concrete Query Contribution is generated. Processing then continues according to method 300 described above.

If the access method is not a composed access method, processing proceeds from step 412 to step 418. Step 418 is representative of any other access methods types contemplated as embodiments of the present invention. However, it should be understood that embodiments are contemplated in which less then all the available access methods are implemented. For example, in a particular embodiment only simple access methods are used. In another embodiment, only simple access methods and filtered access methods are used.

As described above, it may be necessary to perform a data conversion if a logical field specifies a data format different from the underlying physical data. In one embodiment, an initial conversion is performed for each respective access method when building a Concrete Query Contribution for a logical field according to the method 400. For example, the conversion may be performed as part of, or immediately following, the steps 404, 408 and 416. A subsequent conversion from the format of the physical data to the format of the logical field is performed after the query is executed at step 322. Of course, if the format of the logical field definition is the same as the underlying physical data, no conversion is necessary.

OTHER EMBODIMENTS OF DATA REPOSITORY ABSTRACTION COMPONENTS

In one embodiment, a different single data repository abstraction component 148 is provided for each separate physical data representation 214 (as in FIGS. 2B and 2C). In an alternative embodiment, a single data repository abstraction component 148 contains field specifications (with associated access methods) for two or more physical data representations 214. In yet another embodiment, multiple data repository abstraction components 148 are provided, where each data repository abstraction component 148 exposes different portions of the same underlying physical data (which may comprise one or more physical data representations 214). In this manner, a single application 140 may be used simultaneously by multiple users to access the same underlying data where the particular

portions of the underlying data exposed to the application are determined by the respective data repository abstraction component 148. This latter embodiment is described in more detail in co-pending United States Patent Application (Attorney Docket ROC920020088), entitled "DYNAMIC END USER SPECIFIC CUSTOMIZATION OF AN APPLICATION'S PHYSICAL DATA LAYER THROUGH A DATA REPOSITORY ABSTRACTION LAYER" and assigned to International Business Machines Inc.

In any case, a data repository abstraction component 148 contains (or refers to) at least one access method which maps a logical field to physical data. To this end, as illustrated in the foregoing embodiments, the access methods describe a means to locate and manipulate the physical representation of data that corresponds to a logical field.

In one embodiment, the data repository abstraction component 148 is extended to include description of a multiplicity of data sources that can be local and/or distributed across a network environment. The data sources can be using a multitude of different data representations and data access techniques. In one embodiment, this is accomplished by configuring the access methods of the data repository abstraction component 148 with a location specification defining a location of the data associated with the logical field, in addition to the method used to access the data.

Referring now to FIG. 5, a logical/runtime view of an environment 500 having a plurality of data sources (repositories) 502 is shown and illustrates one embodiment of the operation of a data repository abstraction component 148 in such an environment. The data sources 502 to be accessed via the data repository abstraction component 148 may be local, remote or both. In one embodiment, the data sources 502 are representative of the databases 156-157 shown in FIG. 1. In general, the data repository abstraction component 148 is similarly configured to those embodiments described above. As such, the data repository abstraction component 148 has logical field definitions and an associated access method for each logical field definition. However, in contrast to other embodiments in which only a single data source is accessed, the access methods are now configured with location specifications in addition to physical representation specifications. The location specifications describe the location (i.e., the data source) in which the data to be accessed (i.e., the data associated with the logical field definitions) is located. However, in one embodiment, it is contemplated that some access

methods may be configured without location specifications, indicating a default to a local data source.

In general, FIG. 5 shows the application 140, the abstract query specification 142 (also referred to herein as the application query specification), the data repository abstraction component 148 (used to map logical fields to access methods) and the runtime component 150 responsible for converting an abstract query into one or more data access requests supported by the data repositories 502 containing the physical information being queried. In contrast to some embodiments described above, the data repository abstraction component 148 and runtime component 150 of FIG. 5 are configured to support the definition and query of logical fields having associated data that may be distributed across multiple local and/or remote physical data repositories 502 (also referred to herein as local/remote data sources 502) and which may be accessed via a multitude of query-based and procedural based interfaces.

To this end, the application 140 defines its data requirements in terms of the abstract query specification 142 which contains query selection and/or update logic based on logical fields, not the physical location or representation of the actual data involved. The data repository abstraction component 148 comprises logical field definitions 504 and an access method 506 for each logical field. The logical field definitions 504 describe the logical fields available for use by the application 140. In one aspect, the data repository abstraction component 148 governs the information available for use by the application 140. Addition of new logical fields, presented in a new local or remote data source, are thereby made available for use by applications. Each of the access methods 506 define the mapping between a logical field and its physical representation in a local/remote data source 502. This relationship may be understood with reference to FIG. 6.

FIG. 6 shows an illustrative abstract query 602 comprising a plurality of logical fields $604_1 \ldots 604_N$ (collectively the logical fields 604). Each of the logical fields 604 are related (represented by lines 606) to an access method $608_1 \ldots 608_N$ (collectively the access methods 608) by the definition of the particular data repository abstraction component 148. Physical representation information in the access methods 608 includes the name of the access method to be used (here represented as "access method for F1", "access method for F2", etc.) and a plurality of parameters to be passed to the named access method and which describe how to access the physical data associated with the logical field. In

general, such parameters include a locator parameter $610_1 \ldots 610_N$ (collectively the locator parameters 610; also referred to herein as a location specification) and other access parameters needed to access the data. A given data repository abstraction component instance may represent information that is managed by multiple local and remote physical data repositories.

Illustrative embodiments in which a data repository abstraction component instance may be configured with a location specification and other access parameters needed to access the data are shown in FIGS. 7-8. Referring first to FIG. 7, a field specification 700 of a data repository abstraction component configured with a relational access method is shown. The field specification 700 is specific to a particular logical field identified by a field name 702 "CreditRatingDescription" and having an associated access method. The associated access method name 704 is "simple-remote" indicating that the access method is a simple field access method in which the logical fields are mapped directly to a particular entity in the underlying physical data representation and that the data is remotely located. In this case, the logical field is mapped to a given database table "credit_t" and column "desc". The "URL" is the location specification (locator parameter) which specifies the location of the physical data. In this case, the "URL" includes an identifier of a JDBC driver to use, a remote system name holding the data (remotesystem.abc.com) and a database schema containing the data (creditschema). "JDBC Driver" is the name of the Java class that implements SQL access to this type of remote database.

Referring now to FIG. 8, a field specification 800 of a data repository abstraction component configured with a procedural access method is shown. The field specification 800 is specific to a particular logical field identified by a field name 802 "CreditRating" and having an associated access method. The associated access method name 804 is "procedural" indicating that the access method is a procedural access method. "Service Spec" identifies the Web Services Description Language (WSDL) definition for the web service to access. WSDL is a standard interface definition language for Web Services. Web Services is a standard method used to invoke software applications using the established Web infrastructure for communication and using standard data representation technologies such as XML to represent information passed between a calling application and the Web Service that is invoked. "Service Name" identifies the name of the web service to be accessed out of the set of possible services defined within the "Service Spec". "Port

Name" identifies the port name for the service to be accessed out of the set of possible port names defined within "Service Name". The named port defines the network address for the service. "Operation" is the name of the operation to invoke. Web Services can support more than one function referred to as "operations". "Input" identifies input required when invoking a web service. In this case, a last name value is provided as input to the service. "Output" identifies the output data item that is associated with this logical field. Services may return several pieces of output when they are called. Accordingly "Output" identifies defines the piece of output data that is associated with the current logical field.

Note that in the case of procedural access methods, the field specification of a data repository abstraction component for local data may look substantially identical to the field specification 800 shown in FIG. 8 for accessing remote data. The only difference would be that in the local case the referenced WSDL document would have a URL pointing back to the local server the service is running on.

Referring again to FIG. 5, one embodiment of the operation of the runtime component 150 is now described. In general, the runtime component is responsible for building and executing an executable query based on an abstract query. To this end, at block 510, the runtime component 150 parses the abstract query and uses the data repository abstraction component 148 to map references to one or more logical fields to their corresponding physical location and method of access (collectively referred to herein as the access methods 506). In one embodiment, the runtime component 150 partitions (block 512) overall physical data query requirements into groups (referred to as "sub-queries" 514) representing access to the same physical resource using the same method of access. The "sub-queries" are then executed (block 516). Results from each of the sub-queries 514 are combined and normalized (block 518) before the collective query results 520 are returned to the application 140. In one aspect, this query partitioning approach allows the runtime component 150 to run multiple sub-queries in parallel, taking advantage of multi-CPU hardware architectures.

In one embodiment, the runtime component 150 also manages a local data cache 522. The local data cache 522 contains data retrieved for certain logical fields and is used during subsequent queries as a first choice for lookup of logical fields that were identified in the data repository abstraction component as being cache enabled. Logical fields that are advantageously managed in a cached fashion are those whose values

are relatively static and/or which incur significant overhead to access (where overhead is measured in either time required to fetch the data or monetary expense of accessing the data, assuming some information is managed in a pay-per-use model).

In various embodiments, numerous advantages over the prior art are preferably provided. In one aspect, advantages are preferably achieved by defining a loose coupling between the application query specification and the underlying data representation. Rather than encoding an application with specific table, column and relationship information, as is the case where SQL is used, the application defines data query requirements in a more abstract fashion that are then bound to a particular physical data representation at runtime. The loose query-data coupling preferably enables requesting entities (e.g., applications) to function even if the underlying data representation is modified or if the requesting entity is to be used with a completely new physical data representation than that used when the requesting entity was developed. In the case with a given physical data representation is modified or restructured, the corresponding data repository abstraction is preferably updated to reflect changes made to the underlying physical data model. The same set of logical fields are available for use by queries, and have merely been bound to different entities or locations in physical data model. As a result, requesting entities written to the abstract query interface continue to function unchanged, even though the corresponding physical data model may have undergone significant change. In the event a requesting entity is to be used with a completely new physical data representation than that used when the requesting entity was developed, the new physical data model may be implemented using the same technology (e.g., relational database) but following a different strategy for naming and organizing information (e.g., a different schema). The new schema will contain information that may be mapped to the set of logical fields required by the application using simple, filtered and composed field access method techniques. Alternatively, the new physical representation may use an alternate technology for representing similar information (e.g., use of an XML based data repository versus a relational database system). In either case, existing requesting entities written to use the abstract query interface can easily migrate to use the new physical data representation with the provision of an alternate data repository abstraction which maps fields referenced in the query with the location and physical representation in the new physical data model.

In another aspect, the ease-of-use for the application builder and the end-user is facilitated. Use of an abstraction layer to represent logical fields in an underlying data repository enables an application developer to focus on key application data requirements without concern for the details of the underlying data representation. As a result, higher productivity and reduced error rates are achieved during application development. With regard to the end user, the data repository abstraction preferably provides a data filtering mechanism, exposing pertinent data and hiding nonessential content that is not needed by a particular class end-user developing the given query.

Further, the presence of multiple data sources can be used advantageously. By configuring the data repository abstraction components with location specifications, multiple data sources can preferably be accessed, whether the data sources are local or remote. In this manner, an infrastructure is provided which is capable of capitalizing on the distributed environments prevalent today.

Solutions implementing this model use the provided abstract query specification to describe its information requirements, without regard for the location or representation of the data involved. Queries are submitted to the runtime component which uses the data repository abstraction component to determine the location and method used to access each logical piece of information represented in the query. In one embodiment, the runtime component also includes the aforementioned data caching function to access the data cache.

This model preferably allows solutions to be developed, independent of the physical location or representation of the data used by the solution, making it possible to easily deploy the solution to a number of different data topologies and allowing the solution to function in cases where data is relocated or reorganized over time. Preferably this approach also simplifies the task of extending a solution to take advantage of additional information. Extensions are made at the abstract query level and do not require addition of software that is unique for the location or representation of the new data being accessed. This method preferably provides a common data access method for software applications that is independent of the particular method used to access data and of the location of each item of data that is referenced. The physical data accessed via an abstract query may be represented relationally (in an existing relational database system), hierarchically (as XML) or in some other physical data representation model. A multitude of data access

methods are also supported, including those based on existing data query methods such as SQL and XQuery and methods involving programmatic access to information such as retrieval of data through a Web Service invocation (e.g., using SOAP) or HTTP request.

It should be noted that any reference herein to particular values, definitions, programming languages and examples is merely for purposes of illustration. Accordingly, the invention is not limited by any particular illustrations and examples. Further, while aspects of the invention are described with reference to SELECTION operations, other input/output operation are preferably contemplated, including well-known operations such as ADD, MODIFY, INSERT, DELETE and the like. Of course, certain access methods may place restrictions on the type of abstract query functions that can be defined using fields that utilize that particular access method. For example, fields involving composed access methods are not viable targets of MODIFY, INSERT and DELETE.

Claims:

1. A computer-implemented method of providing access to data in an environment of multiple data repositories, comprising: performing an operation by a computer, the operation comprising: providing a data abstraction model comprising a plurality of logical fields for defining an abstract query; and for each of the plurality of logical fields, providing an access method which specifies at least a method for accessing the data and a location of the data, wherein each access method is of a given type according to the particular physical data representation of the data to be accessed; receiving the abstract query from a requesting entity, the abstract query having been composed according to a query specification that provides an interface to the data abstraction model; transforming the abstract query into a query consistent with a particular physical data representation of the data, wherein transforming the abstract query comprises partitioning the abstract query into sub-queries grouped according to the access method types; and accessing a data repository specified by the location in the access method for the physical entity of the data for a particular logical field of the plurality of logical fields.

2. The computer-implemented method of claim 1, where the query consistent with the particular physical data representation is one of a SQL query, an XML query and a procedural request.

3. The computer-implemented method of claim 1, wherein the access method types are selected from a group comprising an SQL query, an XML query and a procedural request.

4. A computer-implemented method of accessing data in an environment of multiple data repositories, comprising: performing en operation by a computer, the operation comprising: receiving, from a requesting entity, an abstract query according to a query specification of the requesting entity; wherein the query specification provides an interface to a data abstraction model comprising a definition for each of a plurality of logical fields of the abstract query, each definition of each logical field defining an access method which maps the logical field to a respective physical entity of the data by defining (i) a method for accessing the respective physical entity and (ii) a location

- 2 NOV 2006

for the respective physical entity; wherein each access method is of a given type according to the particular physical data representation of the data to be accessed; and transforming the abstract query into a query consistent with a particular physical data representation of the data according to the access methods; wherein transforming the abstract query into the query consistent with the particular physical data representation comprises partitioning the abstract query into sub-queries grouped according to the access method types.

5. The method of claim 4, further comprising accessing a data repository specified by the location for a physical entity of the data for a particular logical field of the plurality of logical fields.

6. The method of claim 4, wherein the abstract query comprises at least one selection criterion and a result specification.

7. The method of claim 4 wherein transforming comprises: for a physical entity of the data for a particular logical field of the plurality of logical fields, determining whether the physical entity of the data is located in a local cache; and if not, creating query language of the query configured to access a data repository specified by the location in the access method for the physical entity of the data.

8. The method of claim 4, wherein the access method types are selected from a group comprising an SQL query, an XML query and a procedural request.

9. A computer-readable medium containing a program which, when executed by a processor, performs an operation of accessing data having a particular physical data representation, the operation comprising: receiving, from a requesting entity, an abstract query according to a query specification of the requesting entity; wherein the query specification provides an interface to a data abstraction model comprising a definition for each of a plurality of logical fields of the abstract query, each definition of each logical field defining an access method which maps the logical field to a respective physical entity of the data by defining (i) a method for accessing the respective physical entity and (ii) a location for the respective physical entity; wherein each access method is of a given type according to the particular physical data

representation of the data to be accessed; and transforming the abstract query into a query consistent with a particular physical data representation of the data according to the access methods; wherein transforming the abstract query into the query consistent with the particular physical data representation comprises partitioning the abstract query into sub-queries grouped according to the access method types.

10. The computer-readable medium of claim 9, wherein the operation further comprises accessing a data repository specified by the location for the physical entity of the data for a particular logical field of the plurality of logical fields.

11. The computer-readable medium of claim 9, wherein the abstract query comprises at least one selection criterion and a result specification.

12. The computer-readable medium of claim 9, wherein transforming comprises: for the physical entity of the data for a particular logical field of the plurality of logical fields, determining whether the physical entity of the data is located in a local cache; and if not, creating query language of the query configured to access a data repository specified by the location in the access method for the physical entity of the data.

13. The computer-readable medium of claim 9, wherein the access method types are selected from a group comprising an SQL query, an XML query and a procedural request.

14. A computer, comprising: a memory containing at least: (i) a requesting entity comprising a query specification providing a definition for an abstract query comprising a plurality of logical fields; (ii) a data repository abstraction component comprising mapping rules which map the logical fields to physical entities of data, wherein the mapping rules comprise (a) location specifications for each of at least a portion of the logical fields of the abstract query, and wherein each of the location specifications specify a location of a data source containing a physical entity to be accessed; and (b) access methods each defining a method for accessing the respective physical entity to be accessed, wherein each access method is of a given type according to a particular physical data representation of the data to be accessed; and (iii) a runtime component for transforming the abstract query into a query consistent

- 2 NOV 2006

with the respective particular physical data representation of the physical entities of data according to the mapping rules, wherein transforming comprises partitioning the abstract query into sub-queries grouped according to the access method types; and a processor adapted to execute contents of the memory.

15. The computer of claim 14, wherein a first portion of the data sources specified by the respective location specification are local and a second portion are remote.

16. The computer of claim 14, wherein the access method types are selected from a group comprising an SQL query, an XML query and a procedural request.