Description

METHOD OF ESTABLISHING A DATA MANAGEMENT FEE STRUCTURE BASED ON FINE GRAINED DATA ENTITIES

Technical Field

[001] The present invention generally relates to data processing, and more particularly, to accessing data through a logical framework.

Background Art

[002] Information Technology (IT) encompasses all forms of technology used to create, store, exchange, and use information in its various forms (business data, voice conversations, still images, motion pictures, multimedia presentations, and other forms, including those not yet conceived). While information technology provides significant benefits to users, the providers of such technology often incur substantial costs. As such, it is common for providers to charge a fee for services rendered. For example, many service providers manage information databases and allow fee-based access to the databases. Such databases are particularly prevalent in various professions (e.g., medical, legal, etc.) as well as in research and development. However, such service providers in the information technology industry are providing users with access to databases using relatively inflexible and restrictive fee models.

[003] Consider a service provider who owns and manages a repository of information which they provide fee-based access to. It may be a database of medical research information or any other kind of data that is of value within a particular domain. There are a number of methods established today which govern the fee structure for these kinds of environments. In some cases, the service provider may charge an annual fee for the right to access information in the repository. In other cases, the service provider may charge a per transaction fee for information retrieved. These models are adequate if all data in the repository is considered of equal value, but do not address issues where the value of information can vary between various entities in the repository.

[004] Consider further service providers that are in the data hosting business, i.e., those who provide data storage and management services for a fee. One model that is used today involves a fee structure based on the volume of data stored. In this case, all types of data are considered equal; all types of data are viewed as simply a set of bytes being stored. This model works adequately if there is no distinction between how different types of data are managed, but is not a good solution if there are different types of services provided for different types of information. For example, assume a service provider who is storing patient demographic, account, test result and medical diagnosis information for a given client. There may be different levels of service associated with

each type of information. The service provider may provide services to secure patient authorization to use data for research that would apply to individual patients stored in the repository. The service provider may need to provide more stringent access control and data encryption support for certain types of data. There may also be a set of data analysis services provided for some types of information. Existing fee models do not account for these variations on the type of data stored and services associated with that data.

Disclosure of Invention

[005] According to a first aspect, there is provided a method of providing fee-based access to data, comprising: providing an abstract model for logically defining abstract operations to access the data, the abstract model comprising: (i) a plurality of logical fields; (ii) a mapping rule for each of the plurality of logical fields, which map the plurality of logical fields to physical entities of the data; and (iii) a fee schedule for each of the plurality of logical fields, wherein each fee schedule for a given logical field defines a fee to be charged when the given logical field is involved in an abstract operation to access a physical entity corresponding to the given logical field.

[006] Systems, methods and articles of manufacture for accessing data for a fee are preferably provided. Fee schedules are preferably defined for any arbitrary granularity of data, including for fields and data structures (e.g., tables in a database). Fees may be calculated based on the type of operation to be performed. Fees may also be calculated per operation and/or per data item involved in the operation.

[007] Preferably there is provided a more flexible fee based model for accessing data.

[008] In a particular embodiment physical data is accessed through an abstraction model which defines fee schedules. The abstraction model includes metadata describing and defining a plurality of logical fields. In one embodiment, the metadata also describes associations between sets of logical fields each of which may correspond to (i.e., point to) separate physical entities. The sets of logical fields are referred to as model entities, which facilitate accessing physical data. In some cases, a model entity may also be defined by a single logical field corresponding to a single physical entity. Whether or not model entities are defined, a fee may be associated with one or more logical fields. The fee may be based, for example, on the type of operation, the number of items involved in an operation and/or on the number of requests made with respect to a logical field.

[009] In a preferred embodiment the data is accessed according to an abstract operation comprising at least two of the plurality of logical fields and the fee to be charged is calculated based on separate fee schedules corresponding to each of the at least two plurality of logical fields.

[010] In a preferred embodiment a run-time component is provided configured with transformation instructions to transform each abstract operation, comprising logical fields selected from the plurality of logical fields, into a physical operation consistent with the physical data. Preferably a fee calculator is also provided which is configured to calculate a fee for executing physical operations based on the fee schedules.

[011] In a preferred embodiment the abstract operation is an abstract query comprising at least two logical fields. Preferably a corresponding fee schedule is accessed for each of the at least two logical fields. Preferably a per request fee is determined for a first one of the at least two logical fields, wherein the per request fee is charged for each abstract operation involving the first one of the at least two logical fields. Preferably a per item fee is determined for a second one of the at least two logical fields, wherein the per item fee is charged for each instance of the second one of the at least two logical fields involved in a given abstract operation.

[012] Preferably the per item fee is multiplied by a number of instances of the second one of the at least two logical fields to determine a product, and the product and the per request fee are summed to determine the fee to be charged.

[013] In a preferred embodiment at least one fee schedule defined by the abstract model specifies a first fee for a first type of operation and a second fee for a second type of operation. Preferably the fee to be charged is calculated based on the type of operation performed.

[014] Preferably the first type of operation is a query and the second type of operation is one of an insert and an update.

[015] According to a second aspect, there is provided a method of providing fee-based access to physical data comprising a plurality of physical entities each comprising a plurality of physical fields. The method includes providing an abstract model for defining abstract operation specifications logically describing operations to access the data. The abstract model may include (a) a plurality of logical fields; (b) a mapping rule for each of the plurality of logical fields, which map each of the plurality of logical fields to at least one of the physical entities of the data; (c) a plurality of model entity definitions, each comprising at least one logical field corresponding to a physical field of a physical entity; and (d) a logical field fee schedule for each of the plurality of logical fields, wherein the fee schedules each specify a fee for accessing a corresponding physical field as part of a physical operation.

[016] According to a preferred embodiment, it is possible to transform, according to the abstract model, abstract operation specifications into physical operation specifications consistent with the physical data, wherein each abstract operation specification includes at least one of the plurality of model entity definitions. Preferably a fee is calculated for executing physical operations based on the fee schedules.

[017] According to a preferred embodiment at least one logical field fee schedule defined by the abstract model specifies a first fee for a first type of operation and a second fee for a second type of operation.

[018] Preferably the first type of operation is a query and the second type of operation is one of an insert and an update.

[019] According to a third aspect, there is provided a method of providing fee-based access to data comprising a plurality of physical entities, each comprising a plurality of physical fields. The method includes receiving instructions to perform an operation for accessing the data; performing the operation; determining field-specific fees for each of a plurality of the physical fields accessed by the operation; and calculating a total fee to be charged to a user for the operation.

[020] The physical entities may be database tables.

[021] The operation may be one of a query, an insert and an update.

[022] According to a preferred embodiment, determining field-specific fees comprises determining whether a field-specific fee is a per request fee or a per item fee, wherein the per request fee is a singular fee charged for the operation regardless of a number of instances a corresponding physical field is included in the operation and wherein the per item fee is charged for each instance of a corresponding physical field included in the operation.

[023] According to a preferred embodiment, determining field-specific fees comprises accessing fee schedules for each respective physical field accessed by the operation.

[024] Preferably each of the fee schedules defines a separate fee for each separate operation type.

[025] Preferably the separate operation types comprise queries, inserts and updates.

[026] According to a preferred embodiment, determining field-specific fees comprises accessing an abstract model for logically defining the operation accessing the data. Preferably the abstract model comprises: (i) a plurality of logical fields; (ii) a mapping rule for each of the plurality of logical fields, which map the plurality of logical fields to physical entities of the data; and (iii) a fee schedule for each of the plurality of logical fields, wherein each fee schedule for a given logical field defines a fee to be charged when the given logical field is involved in an abstract operation to access a physical entity corresponding to the given logical field.

[027] Preferably each mapping rule comprises an access method for each logical field of an abstract operation specification logically defining the operation accessing the data, the access method describing a physical location of a physical entity.

[028] According to a preferred embodiment, each fee schedule defines at least one of a per request fee and a per item fee, wherein the per request fee is a singular fee charged for the operation regardless of a number of instances a corresponding logical field is

included in the operation and wherein the per item fee is charged for each instance of a corresponding logical field included in the operation.

[029] According to a preferred embodiment, it is possible to transform, according to the abstract model, abstract operations into physical operation consistent with the physical data, wherein each abstract operation includes at least one of the plurality of model entity definitions. Preferably it is also possible to calculate the fee for executing physical operations based on the fee schedules.

[030] According to a preferred embodiment, data is accessed according to the abstract operation, the abstract operation comprising at least two of the plurality of logical fields. Preferably the fee is calculated based on separate fee schedules corresponding to each of the at least two plurality of logical fields.

[031] According to a preferred embodiment, there is provided a run-time component configured with transformation instructions to transform each abstract operation, comprising logical fields selected from the plurality of logical fields, into a physical operation consistent with the physical data. Preferably there is also provided a fee calculator configured to perform the calculating of the fee for executing physical operations based on the fee schedules.

[032] According to a preferred embodiment the abstract operation is an abstract query comprising at least two logical fields and a corresponding fee schedule is accessed for each of the at least two logical fields. Preferably a per request fee is determined for a first one of the at least two logical fields, wherein the per request fee is charged for each abstract operation involving the first one of the at least two logical fields. Preferably a per item fee is determined for a second one of the at least two logical fields, wherein the per item fee is charged for each instance of the second one of the at least two logical fields involved in a given abstract operation.

[033] Preferably the per item fee is multiplied by a number of instances of the second one of the at least two logical fields to determine a product, and the product and the per request fee are summed to determine the fee to be charged.

[034] According to a preferred embodiment, at least one fee schedule defined by the abstract model specifies a first fee for a first type of operation and a second fee for a second type of operation. Preferably the fee is calculated to be charged based on the type of operation performed.

[035] Preferably the first type of operation is a query and the second type of operation is one of an insert and an update.

[036] According to a fourth aspect, there is provided a computer-readable medium containing a program which, when executed by a processor, performs operations for accessing physical data comprising a plurality of physical entities, each having a plurality of physical fields. The operation includes receiving instructions to perform an

operation accessing the data; causing performance of the operation; determining field-specific fees for each of a plurality of the physical fields accessed by the operation; and calculating a total fee to be charged to a user for the operation.

[037] The physical entities may be database tables.

[038] The operation may be one of a query, an insert and an update.

[039] According to a preferred embodiment, determining field-specific fees comprises determining whether a field-specific fee is a per request fee or a per item fee, wherein the per request fee is a singular fee charged for the operation regardless of a number of instances of a corresponding physical field are included in the operation and wherein the per item fee is charged for each instance of a corresponding physical fee included in the operation.

[040] According to a preferred embodiment, determining field-specific fees comprises accessing fee schedules for each respective physical field accessed by the operation.

[041] Preferably each of the fee schedules defines a separate fee for each separate operation type.

[042] Preferably the separate operation types comprise queries, inserts and updates.

[043] According to a preferred embodiment determining field-specific fees comprises accessing an abstract model for logically defining the operation accessing the data. Preferably the abstract model comprises: (i) a plurality of logical fields; (ii) a mapping rule for each of the plurality of logical fields, which map the plurality of logical fields to physical entities of the data; and (iii) a fee schedule for each of the plurality of logical fields, wherein each fee schedule for a given logical field defines a fee to be charged when the given logical field is involved in an abstract operation to access a physical entity corresponding to the given logical field.

[044] Preferably each mapping rule comprises an access method for each logical field of an abstract operation specification logically defining the operation accessing the data, the access method describing a physical location of a physical entity.

[045] According to a fifth aspect, there is provided provides a method for constructing abstract queries defined by a plurality of logical fields which map to a plurality of physical entities of physical data having a particular physical data representation in a database. The method includes receiving user input via a user interface, the input comprising a reference to a model entity definition comprising: (i) two or more logical fields each corresponding to a separate physical entity; and (ii) a fee schedule for accessing physical entities based on the model entity definition. Based on the model entity definition, at least one of the two or more logical fields is selectively added to an abstract query. The method further includes receiving a plurality of abstract query contributions for the abstract query, wherein the plurality of abstract query contributions are defined by selected logical fields and a corresponding value for each of the selected

logical fields; and receiving a plurality of result fields for the abstract query are received, wherein the plurality of result fields is defined by selected logical fields. The abstract query is then converted into a physical query consistent with the particular physical data representation of the data. The physical query is then executed and, on the basis of the fee schedule, a fee to charge for execution of the physical query is calculated.

[046] The plurality of physical entities may be a table in a database.

[047] According to a preferred embodiment, selectively adding the at least one of the two or more logical fields comprises: determining whether the at least one logical field is a required field as specified by the model entity definition; and if so, adding the at least one logical field to the abstract query.

[048] According to a preferred embodiment, a per request fee is defined for a first one of the two or more logical fields and a per result fee is defined for a second one of the two or more logical fields and wherein calculating the fee comprises: (i) calculating a product of the per result fee and a number of results for the second one of the two or more logical fields; and (ii) summing the product and the per request fee.

[049] According to a preferred embodiment, converting the abstract query into the physical query comprises mapping each of the logical fields of the abstract query to respective physical entities of the physical data.

[050] According to a sixth aspect, there is provided a method for modifying physical data comprising a plurality of physical entities and having a particular physical data representation in a database. The method includes receiving a selection of an abstract modification operation; receiving a selection of a model entity definition on which to perform the abstract modification operation, the model entity definition comprising two or more logical fields each corresponding to a separate physical entity; based on at least the received selections, generating at least two physical modification statements, each modifying one of the two separate physical entities of the physical data; ordering the at least two physical modification statements; executing modification operations according to the physical modification statements, whereby the data is modified; and calculating a fee to charge for executing the modification operations based on a defined fee schedule for the model entity definition.

[051] According to a preferred embodiment, the ordering is done according to a physical entity relationships specification defining hierarchical relationships between physical entities of the data to ensure integrity of the data.

[052] According to a preferred embodiment, building the at least two physical modification operations is done according to mapping rules which map the two or more logical fields to their corresponding physical entity.

[053] According to a seventh aspect, there is provided a method of providing a logical

framework for defining abstract operations for accessing physical data comprising a plurality of physical entities each comprising a plurality of physical fields, the method including providing an abstract model for defining abstract operation specifications logically describing operations to access the data. The abstract model includes (a) a plurality of logical fields; (b) a mapping rule for each of the plurality of logical fields, which map each of the plurality of logical fields to at least one of the physical entities of the data; (c) a plurality of model entity definitions, each comprising at least one logical field corresponding to a physical field of a physical entity; and (d) model entity fee schedules for each of the plurality of model entity definitions, wherein the fee schedules each specify a fee for accessing a physical field of the corresponding model entity definition. The method further includes providing a run-time component to transform, according to the abstract model, abstract operation specifications into physical operation specifications consistent with the physical data, wherein each abstract operation specification includes at least one user-selected model entity definitions of the plurality of model entity definitions.

[054] The plurality of physical entities may be a table in a database.

[055] According to a preferred embodiment, each physical operation specification is selected from one of an insert statement and an update statement and wherein the model entity fee schedules define different fees for each statement.

[056] According to a preferred embodiment, each physical operation specification is a query, and the model entity fee schedules define fees specific to queries.

[057] According to a preferred embodiment, the abstract operation specification is an abstract query and it is possible to receive, via a user interface, the abstract query comprising a plurality of query conditions, result fields and a selection of one of the model entity definitions. Preferably it is possible to access the model entity definition corresponding to the selection and to determine whether the model entity definition corresponding to the selection specifies one or more required result fields and if so, the one or more required result fields are added to the query.

[058] According to a preferred embodiment, it is possible for the run-time component to transform according to the abstract model, a single abstract operation specification into at least two separate physical operation specifications consistent with the physical data, wherein each physical operation specification modifies a different physical entity of the data and wherein each physical operation specifications is ordered for execution according to a physical entity relationships specification defining hierarchical relationships between the physical entities of the data.

[059] According to a preferred embodiment, a requesting entity issues a request to execute a single abstract operation specification and the run-time component transforms the single abstract operation specification into the at least two physical

operation specifications for modifying the data.

[060] Preferably transforming the single abstract operation specification into the at least two physical operation specifications comprises: generating the at least two physical operation specifications; and ordering the at least two physical operation specifications according to a physical entity relationships specification of the abstract model.

[061] According to a preferred embodiment each mapping rule comprises an access method for one of the plurality of logical fields.

[062] Preferably the access method describes a location of the physical entities of the data.

[063] According to a preferred embodiment, there is provided a logical field fee schedule for each of the plurality of logical fields, wherein the fee schedules each specify a fee for accessing a corresponding physical field as part of a physical operation specification.

[064] Preferably at least one fee schedule defined by the abstract model specifies a first fee for a first type of operation and a second fee for a second type of operation.

[065] Preferably the first type of operation is a query and the second type of operation is one of an insert and an update.

[066] According to an eighth aspect, there is provided a computer-readable medium containing a program which, when executed by a processor, provides a logical framework for defining abstract query operations. The program includes an abstract model for defining abstract queries logically describing operations to query the data, the abstract model including (i) a plurality of logical fields; (ii) a mapping rule for each of the plurality of logical fields, which map the plurality of logical fields to physical entities of the data; and (iii) a fee schedule for each of the plurality of logical fields. The program further includes a run-time component configured with transformation instructions to transform an abstract query, comprising logical fields selected from the plurality of logical fields, into a physical query consistent with the physical data; and a fee calculator configured to calculate a fee for executing physical queries based on the fee schedules.

[067] Preferably the mapping rules comprise an access method for each of the plurality of logical fields, wherein the access method describes a location of the physical entities of the data.

[068] According to a ninth aspect, there is provided a computer comprising a memory and at least one processor, and further comprising a logical framework for defining abstract modification operations for modifying physical data, the logical framework including an abstract model for defining an abstract modification specification logically describing an operation to modify the data. The abstract model includes (i) a plurality of logical fields; (ii) a mapping rule for each of the plurality of logical fields, which

map the plurality of logical fields to physical entities of the data; and (iii) a fee schedule for each of the plurality of logical fields. The logical framework further includes a run-time component to transform an abstract query, comprising logical fields selected from the plurality of logical fields, into a physical query consistent with the physical data; and a fee calculator configured to calculate a fee for executing physical queries based on the fee schedules.

[069] According to a preferred embodiment, each fee schedule defines at least one of a per request fee and a per item fee, wherein the per request fee is a singular fee charged for the operation regardless of a number of instances a corresponding logical field is included in the operation and wherein the per item fee is charged for each instance of a corresponding logical field included in the operation.

[070] According to a preferred embodiment, the mapping rules comprise an access method for each of the plurality of logical fields, wherein the access method describes a location of the physical entities of the data.

[071] According to a tenth aspect, there is provided a method for providing fee-based access to data comprising a plurality of physical entities, each comprising a plurality of physical fields, the method comprising receiving, via a user interface, user input comprising instructions for an operation for accessing the data selected fields of the plurality of the physical fields; determining field-specific fees for each of the selected fields; calculating a fee to be charged to a user for accessing the selected fields; and displaying the fee to the user via a user interface.

[072] The physical entities may be database tables.

[073] According to a preferred embodiment, determining field-specific fees comprises determining whether a field-specific fee is a per request fee or a per item fee, wherein the per request fee is a singular fee charged for the operation regardless of a number of instances a corresponding physical field is included in the operation and wherein the per item fee is charged for each instance of a corresponding physical field included in the operation.

[074] According to a preferred embodiment, determining field-specific fees comprises accessing fee schedules for each respective physical field accessed by the operation.

[075] Preferably each of the fee schedules defines a separate fee for each separate operation type.

[076] Preferably the separate operation types comprise queries, inserts and updates.

[077] According to an eleventh aspect, there is provided a method for displaying fee information for fee-based access to data comprising a plurality of physical entities, each comprising a plurality of physical fields. The method comprises displaying one or more user interface screens for construction of queries; receiving, via the one or more user interface screens, user input defining a query configured to access selected fields

of the plurality of physical fields; and displaying, via the one or more user interface screens, a field-specific access fee for each of the selected fields.

[078] Preferably it is possible to display a per query fee, the per query fee being a singular fee charged for the query regardless of a number of instances a corresponding physical field is included in the query.

[079] According to a preferred embodiment, it is possible to determine the field-specific fees by accessing an abstract model for logically defining the query. Preferably the abstract model comprises: (i) a plurality of logical fields; (ii) a mapping rule for each of the plurality of logical fields, which map the plurality of logical fields to physical entities of the data; and (iii) a fee schedule for each of the plurality of logical fields, wherein each fee schedule for a given logical field defines a fee to be charged when the given logical field is involved in a query to access a physical entity corresponding to the given logical field.

[080] Preferably each mapping rule comprises an access method for each logical field of an abstract operation specification logically defining the operation accessing the data, the access method describing a physical location of a physical entity.

[081] According to a preferred embodiment, each fee schedule defines at least one of a per request fee and a per item fee, wherein the per request fee is a singular fee charged for the query regardless of a number of instances a corresponding logical field is included in the query and wherein the per item fee is charged for each instance of a corresponding logical field included in the query.

[082] It will be appreciated that the invention can be implemented in computer software.

Brief Description of the Drawings

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

[084] FIGURE 1 is a block diagram of an illustrative computer architecture in accordance with a preferred embodiment of the present invention.

[085] FIGURE 2 is a relational view of software components of one embodiment of the invention configured to process queries against a physical data source through an abstract representation of the physical data source.

[086] FIGURE 3 is a flow chart illustrating the operation of a runtime component in accordance with a preferred embodiment of the present invention.

[087] FIGURE 4 is a flow chart illustrating the operation of a runtime component in accordance with a preferred embodiment of the present invention.

[088] FIGURE 5 is a view of an abstraction layer, in accordance with a preferred embodiment of the present invention, defining model entities having physical counterparts in a physical data source.

[089] FIGURE 6 is an illustrative model entity specification for a Patient entity and a Test entity.

[090] FIGURE 7 illustrates the augmentation of result fields of an initial abstract query according to a specified model entity.

[091] FIGURES 8-12 are illustrative user interface screens for creating an abstract query using model entities.

[092] FIGURE 13 is a flow chart for augmenting query result fields according to a specified model entity in accordance with a preferred embodiment of the present invention.

[093] FIGURE 14 illustrates the generation of a physical query based on the specification of a model entity in accordance with a preferred embodiment of the present invention.

[094] FIGURE 15 is a flow chart of a method for generating a physical query based on the specification of a model entity in accordance with a preferred embodiment of the present invention.

[095] FIGURE 16 illustrates the use of a model entity to represent a set of query result fields in accordance with a preferred embodiment of the present invention.

[096] FIGURE 17 is a flow chart of a method for expanding the result fields of an abstract query based on the specification of a model entity in accordance with a preferred embodiment of the present invention.

[097] FIGURE 18 is a simplified view of the environment shown in FIGURE 5, and illustrating the modification of a data source by abstract modification operations constructed based on model entities in accordance with a preferred embodiment of the present invention

[098] FIGURE 19 is a flow chart illustrating the generation of an abstract insert specification used to implement an insert operation against physical data in accordance with a preferred embodiment of the present invention.

[099] FIGURES 20A-B are flow charts illustrating the determination of required and optional fields in an insert operation in accordance with a preferred embodiment of the present invention.

[100] FIGURE 21 is a flow chart illustrating the conversion of an abstract insert operation to a physical insert operation in accordance with a preferred embodiment of the present invention.

[101] FIGURE 22 is a flow chart illustrating sorting of an insert statement list to ensure a proper order of execution in accordance with a preferred embodiment of the present invention.

[102] FIGURE 29 is a flow chart illustrating the generation of selection logic for an abstract update operation in accordance with a preferred embodiment of the present invention.

[103] FIGURE 27 is a flow chart illustrating generation of an abstract delete specification used to implement a delete operation against physical data delete operation in accordance with a preferred embodiment of the present invention.

[104] FIGURE 28 is a flow chart illustrating the conversion of an abstract delete operation to a physical delete operation in accordance with a preferred embodiment of the present invention.

[105] FIGURE 30 is a flow chart illustrating sorting of a delete statement list to ensure a proper order of execution in accordance with a preferred embodiment of the present invention.

[106] FIGURE 31 is the abstraction layer of FIGURE 5 additionally configured with fee schedules in accordance with a preferred embodiment of the present invention.

[107] FIGURE 32 is a detailed view of an illustrative data model.

[108] FIGURE 33 is an illustrative model entity specification for a Patient entity and a Test entity.

[109] FIGURES 34A-B (collectively referred to as FIGURE 34) are flow charts illustrating fee-based query and insert operations in accordance with a preferred embodiment of the present invention.

[110] FIGURES 35A-B (collectively referred to as FIGURE 35) are flow charts illustrating fee-based update operations in accordance with a preferred embodiment of the present invention.

[111] FIGURE 36-41 are a series of graphical user interfaces for creating a query and displaying a calculated fee for the query in accordance with a preferred embodiment of the present invention.

Mode for the Invention

[112] INTRODUCTION

[113] Systems, methods and articles of manufacture for accessing data for a fee are provided. Fee schedules are defined for any arbitrary granularity of data, including for fields and data structures (e.g., tables in a database). In one embodiment, fees are calculated based on the type of operation to be performed. Fees may also be calculated per operation and/or per data item involved in the operation.

[114] In a particular embodiment, physical data is accessed through an abstraction model which defines fee schedules corresponding to the physical data accessed. In one embodiment, a particular data definition framework (also referred to herein as a data abstraction model (DAM)) is provided for accessing and modifying data independent of the particular manner in which the data is physically represented. The data may be located in a single repository (i.e., source) of data or a plurality of different data repositories. Thus, the DAM may provide a logical view of one or more underlying

data repositories. By using an abstract representation of a data repository, the underlying physical representation can be more easily changed or replaced without affecting the application making the changes. Instead, the abstract representation is changed with no changes required by the application. In addition, multiple abstract data representations can be defined to support different applications against the same underlying database schema that may have different default values or required fields.

[115] The DAM includes metadata describing and defining a plurality of logical fields. The metadata also describes associations between sets of logical fields. The sets of logical fields are referred to as model entities. The model entities are used to access data through queries and modification operations. In one aspect, model entities define required output fields for queries involving the model entity. In another aspect, model entities establish a focal point within the data environment that can be used to implement the appropriate logic needed to correlate information that spans multiple physical entities (e.g. tables) in the underlying physical data repository. In still another aspect, model entities define a minimal set of fields used to derive the complete set of fields involved in data insert and delete operations for an instance of the model entity.

[116] In one embodiment, the DAM includes fee schedules for individual logical fields, categories of logical fields or the model entities. The fee schedules may be structured, for example, according to the operation to be performed (e.g., query, insert, update, delete) and on a per item or per request basis.

[117] One embodiment of the invention is implemented as a program product for use with a computer system 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 specifically includes information downloaded from the Internet and other networks. Such signal-bearing media, when carrying computer-readable instructions that direct the functions of the present invention, represent embodiments of the present invention.

[118] 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 present invention 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. In addition, various programs described hereinafter may be identified based upon the application for which they are implemented in a specific embodiment of the invention. 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.

[119] PHYSICAL VIEW OF ENVIRONMENT

[120] 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 (i.e., generally any requesting entity such as a user or application) computer 102 (three such client computers 102 are shown) and at least one server computer 104 (one such server computer 104 is shown). 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. However, it is noted that aspects of the invention need not be implemented in a distributed environment. As such, the client computers 102 and the server computer 104 are more generally representative of any requesting entity (such as a user or application) issuing queries and a receiving entity configured to handle the queries, respectively.

[121] The client computer 102 includes a Central Processing Unit (CPU) 110 connected via a bus 130 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.

[122] 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).

[123] 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.

[124] The memory 112 is preferably a random access memory sufficiently large to hold the necessary programming and data structures 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.

[125] 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.

[126] 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.

[127] 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 necessary programming and data structures that are located on the server computer 104.

[128] 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.

[129] The memory 132 further includes one or more applications 140 and an abstract query interface 146. The applications 140 and the abstract operation 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 operation interface 146 cause the computer system 100 to perform the steps necessary to execute steps or elements embodying the various aspects 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. Illustrative sources against which queries may be issued include local

databases $156_1 \ldots 156_N$, and remote databases $157_1 \ldots 157_N$, 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 operations) or according to an XML schema (accessible by XML operations). 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 which is described by a data definition framework such as the DAM described herein.

[130] In one embodiment, the operations (e.g., queries, inserts, updates, deletes) issued by the applications 140 are defined according to an application operation specification 142 included with each application 140. The operations 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 operations (referred to herein as "abstract operations ") are composed using logical fields defined by the abstract operation interface 146. In particular, the logical fields used in the abstract operations are defined by a data abstraction model 148 of the abstract operations interface 146. The abstract operations are executed by a runtime component 150 which transforms the abstract operations into a form consistent with the physical representation of the data contained in one or more of the databases 156-157. The operations may be configured to access the data and return results (i.e., query the data), or to modify (i.e., insert, delete or update) the data. The application operation specification 142 and the abstract operation interface 146 are further described with reference to FIGS. 2A-B.

[131] In one embodiment, the server computer 104 is further configured with a fee calculator 151. Illustratively, the fee calculator 151 is shown as part of the runtime component 150, although it may alternatively be a separate component. In any case, the fee calculator 151 is invoked to calculate the cost of an operation, such as a query, insert, update or delete.

[132] In one embodiment, elements of an abstract operation 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 152 (e.g., a web server) adapted to service requests from the client computer 102. For example, the process 152 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 the 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 abstraction model 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.

[133] 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 152 shown in FIG. 1 is merely illustrative and other embodiments adapted to support any known and unknown protocols are contemplated.

[134] LOGICAL/RUNTIME VIEW OF ENVIRONMENT

[135] FIGS. 2A-B show a plurality of interrelated components of the invention. By way of illustration, the embodiment represented by FIGS. 2A-B (as well as FIGS. 3-4) is described with reference to queries. However, more generally, the operation to be performed may be any operation affecting data (i.e., insert, delete, update). The requesting entity (e.g., one of the applications 140) issues a query 202 as defined by the respective application operation 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 operation specification 142 may include 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.

[136] The logical fields specified by the application operation specification 142 and used

to compose the abstract query 202 are defined by the data abstraction model 148. In general, the data abstraction model 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.

[137] In general, the data abstraction model 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, 212_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.

[138] 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.

[139] It is noted that the data abstraction model 148 shown in FIG. 2B is merely illustrative of selected logical field specifications and is not intended to be comprehensive. As such, the abstract query 202 shown in FIG. 2B includes some logical fields for which specifications are not shown in the data abstraction model 148, such as "State" and "Street".

[140] 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 must 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.

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

[142] 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.

[143] TABLE I - QUERY EXAMPLE

[144] 001 <?xml version="1.0"?>

[145] 002 <!--Query string representation: (FirstName = "Mary" AND LastName =

[146] 003 "McGoon") OR State = "NC"-->

[147] 004 <QueryAbstraction>

[148] 005 <Selection>

[149] 006 <Condition internalID="4">

[150] 007 <Condition field="FirstName" operator="EQ" value="Mary"

[151] 008 internalID="1"/>

[152] 009 <Condition field="LastName" operator="EQ" value="McGoon"

[153] 010 internalID="3" relOperator="AND"></Condition>

[154] 011 </Condition>

[155] 012 <Condition field="State" operator="EQ" value="NC" internalID="2"

[156] 013 relOperator="OR"></Condition>

[157] 014 </Selection>

[158] 015 <Results>

[159] 016 <Field name="FirstName"/>

[160] 017 <Field name="LastName"/>

[161] 018 <Field name="State"/>

[162] 019 </Results>

[163] 020 </QueryAbstraction>

[164] 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.

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

[166] TABLE II - DATA REPOSITORY ABSTRACTION EXAMPLE

[167] 001 <?xml version="1.0"?>

[168] 002 <DataRepository>

[169] 003 <Category name="Demographic">

[170] 004 <Field queryable="Yes" name="FirstName" displayable="Yes">

[171] 005 <AccessMethod>

[172] 006 <Simple columnName="f_name" tableName="contact"></Simple>

[173] 007 </AccessMethod>

[174] 008 <Type baseType="char"></Type>

[175] 009 </Field>

[176] 010 <Field queryable="Yes" name="LastName" displayable="Yes">

[177] 011 <AccessMethod>

[178] 012 <Simple columnName="l_name" tableName="contact"></Simple>

[179] 013 </AccessMethod>

[180] 014 <Type baseType="char"></Type>

[181] 015 </Field>

[182] 016 <Field queryable="Yes" name="State" displayable="Yes">

[183] 017 <AccessMethod>

[184] 018 <Simple columnName="state" tableName="contact"></Simple>

[185] 019 </AccessMethod>

[186] 020 <Type baseType="char"></Type>

[187] 021 </Field>

[188] 022 </Category>

[189] 023 </DataRepository>

[190] Note that lines 004-009 correspond to the first field specification 208_1 of the DAM 148 shown in FIG. 2B and lines 010-015 correspond to the second field specification 208_2. For brevity, the other field specifications defined in Table I have not been shown in FIG. 2B. Note also that Table I illustrates a category, in this case "Demographic". A category is a grouping of one or more logical fields. In the present example, "First Name", "Last Name" and "State" are logical fields belonging to the common category, "Demographic".

[191] 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.

[192] 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.

[193] 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.

[194] 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.

[195] 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.

[196] 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.

[197] OTHER EMBODIMENTS OF DATA REPOSITORY ABSTRACTION COMPONENTS

[198] In one embodiment, a different single data abstraction model 148 is provided for each separate physical data representation 214. In an alternative embodiment, a single data abstraction model 148 contains field specifications (with associated access methods) for two or more physical data representations 214. In yet another embodiment, multiple data abstraction models 148 are provided, where each data abstraction model 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 abstraction model 148. This latter embodiment is described in more detail in United States Patent Application No. 10/132,228, 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 (publication number US20030208486).

[199] In any case, a data abstraction model 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 access methods are further configured with a location specification defining a location of the data associated with the logical field. In this way, the data abstraction model 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 this manner, an infrastructure is provided which is capable of capitalizing on the distributed environments prevalent today. One approach for accessing a multiplicity of data sources is described in more detail in United States Patent Application No. 10/131,984, entitled "REMOTE DATA ACCESS AND INTEGRATION OF DISTRIBUTED DATA SOURCES THROUGH DATA SCHEMA AND QUERY ABSTRACTION" and assigned to International Business Machines, Inc (publication number US20030208458).

[200] In various embodiments, numerous advantages over the prior art are provided. In one aspect, advantages are achieved by defining a loose coupling between the application operation 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 operation requirements in a more abstract fashion that are then bound to a particular physical data representation at runtime. The loose operation-data coupling of the present invention 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 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 operation interface continue to function unchanged, even though the corresponding physical data model has undergone significant change. In the event a requesting entity is to be used with a completely new physical data representation different 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. Al-

ternatively, 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 operation 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.

[201] 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 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.

[202] Solutions implementing the present model use the provided abstract operation specification to describe its information requirements, without regard for the location or representation of the data involved. Operations, e.g., queries, are submitted to the runtime component which uses the data abstraction model to determine the location and method used to access each logical piece of information represented in the query.

[203] In one aspect, this model 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. In another aspect, 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 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.

[204] MODEL ENTITIES

[205] Aspects of the present invention provide data abstraction model entities that serve

to identify a higher level abstraction of the underlying data by representing a composite of individual logical fields. Model entities provide end users and applications a higher level conceptual view of the underlying data that can simplify data query and modification tasks (i.e., insert and deletion). Rather than having to understand all of the individual fields that make up entities such as a patient or a lab test result, the user/application can work at the more conceptual model entity level. As will be described below in more detail, the definition of a model entity contains sufficient metadata to streamline and simplify transactions performed against instances of a model entity.

[206] In the current embodiment, model entities are defined via additional metadata to that already found in an abstract data model representation (i.e., the DAM). More generally, however, model entities can be defined within an abstract data model definition or could be defined external to an abstract data model definition.

[207] Further, embodiments are described with reference to relational databases. However, the invention is applicable to any other data representation including, for example, markup languages such as XML.

[208] Referring now to FIGURE 5, an environment 500 includes a representative data abstraction model (DAM) 502 configured to support accesses (i.e., queries and modification operations) of a physical data source. By way of illustration only, the physical data source being accessed via the data abstraction model 502 is a relational data source 504 containing a plurality of tables 520-523. However, as described above, any data type is contemplated.

[209] The data abstraction model 502 generally includes a plurality of categories 508_{1-5}, a plurality of logical fields specifications 510_{1-16}, a model entity specification 525 and a physical entity relationship specification 526. The categories 508 may be defined for a single logical field or, more commonly, relate two or more logical field specifications 510. The logical fields specifications 510 includes the metadata described above with respect to FIGURE 2, which is not shown for simplicity. Some aspects of the logical fields specifications described above with respect to FIGURE 2 are shown in a simplified form. For example, reference to logical fields used in defining composed fields is represented by arrows, such as in the case of the "Age" logical field specification 510_5 and the "Days to Payment " logical field specification 510_{16}.

[210] In addition, logical fields specifications 510 include supplemental metadata used to implement aspects of the invention. For example, selected logical fields are configured with various attributes including a "required" attribute 514, a "generate" attribute 516 and a "default" attribute 518. Illustratively, the "First Name" logical field specification 510_2, the "Last Name" logical field specification 510_3, the "City" logical field specification 510_8, the "State" logical field specification 510_9, the "Postal Code" logical

field specification 510_{10}, the "Glucose Test" logical field specification 510_{12}, the "Account Number" logical field specification 510_{13} and the "Balance" logical field specification 510_{14} are configured with the "Required" attribute 514. The "Patient ID" logical field specification 510_1 is configured with the "Generate" attribute 516 and the Test Date logical field specification 510_{11} is configured with the "Default Value" attribute 518, where the default value is specified as "Current_Date".

[211] The model entity specification 525 defines a plurality of model entities 506_{1-3} (illustratively three are shown; however, any number of model entities may be defined). Each model entity has a name. Illustratively, a "Patient" model entity 506_1, a "Test" model entity 506_2 and an "Account" model entity 506_3 are defined by the DAM 502.

[212] By way of illustration, additional details of the Patient and Test model entities 506_{1-2} are now described with reference to FIGURE 6. Although not shown, the details of the "Account" model entity 506_3 may have a similar composition. In addition to a name 602, each model entity defines multiple sets of fields used to implement query, insert and delete operations against the physical data corresponding to the model entity. Specifically, each model entity 506 is partitioned to include a query portion 604_{1-2}, the insert portion 606_{1-2} and a delete portion 608_{1-2}. The appropriate portion is accessed according to the type of operation being run against the model entity 506. Note that for queries, the full complement of fields defining a model entity (e.g., Patient) is specified, while in the case of inserts and deletes a subset of all the fields defining the model entity is specified. As will be described in more detail below, the subset of fields include a "seed" field for each corresponding physical entity of a model entity. In any case, it should be clear that a portion of a model entity 506 may include only a single logical field pointing to a single physical entity. Further, a model entity 506 may itself only have a single logical field pointing to a single physical entity. The model entities provide a particular advantage, however, when they span multiple fields/entities since in this case users are able to work with a singular abstract representation rather than being burdened with knowing what logical fields make up an abstract entity. In this regard, it is noted that, in practice, each portion (query, insert and delete) of a model entity 506 is itself a model entity in that the portions each define an abstract entity for a given operation, whether the abstract entity spans multiple logical fields and/or multiple physical fields.

[213] In addition to the model entity metadata, aspects of the invention are implemented by the physical entity relationships specification 526, which is now described with reference to FIGURE 5. The physical entity relationships specification 526 defines the hierarchical relationships between entities in a physical model (i.e., the relational database 520). By way of illustration, the physical entity relationships specification

526 shown in FIGURE 5 relates the patient information table 520 to each of the other tables 521-523 in the data source 504. In each case, the patient information table 520 is primary with respect to a secondary table. Although not illustrated in the physical entity relationships specification 526 of FIGURE 5, it is contemplated that additional levels of hierarchy may be defined. For example, the address information table 521 may be defined as a primary entity with respect to some other secondary table (referred to for convenience as "Table A") not shown in FIGURE 5. In this case, a three-tiered hierarchy is defined in which the patient information table 520 is the primary entity, the address information table 521 is the secondary entity, and Table A is the tertiary entity. In such an arrangement, the patient information table 520 and the address information table 521, and the address information table 521 and Table A are explicitly in a primary-secondary relationship, and by syllogism, the patient information table 520 and Table A are in a primary-secondary relationship.

[214] The physical entity relationships specification 526 also indicates the basis for a primary-secondary relationship between entities. Specifically, the field (i.e., column) on which the relationship is based is specified in brackets [ ]. In the present illustration, the entity relationships are defined for the patient identifier ("ID" and "PID", respectively). Although only one field name is shown specified for each entity, two or more may be specified such that each entity is related by two or more pairs of fields. Consider the following example of a relationship: Entity 1 [field 1, field 3, field 6] à Entity 2 [field 2, field 3, field 4]. In this example, the fields 1, 3 and 6 of the primary entity, Entity 1, are related to fields 2, 3 and 4, respectively, of the secondary entity, Entity 2.

[215] The physical entity relationships specification 526 also specifies whether a relationship between two entities is one-to-one, one-to-many many-to-one or many-to-many. This relationship is specified in parentheses () for each entity. For example, the entities "PatientInfo" and "AddressInfo" are in a one-to-one relationship, while the entities "PatientInfo" and "TestInfo" are in a one-to-many relationship.

[216] The DAM 502 allows a requesting entity 512 (e.g., application 140 of FIGURE 1) to access the data source 504 by issuing a request for results from the data source 504 or by issuing a request to modify data in the data source 504. Generally, both of these requests may be referred to as "queries". However, for convenience, only a request for results will be referred to as a query in the following description.

[217] Query Operations Using Model Entities

[218] In the case of query operations, a set of fields defined by the model entity 506 in the query portion 604 serves a variety of purposes. First, the query portion 604 specifies those fields that are required output from queries involving the model entity. Required fields for query results are identified in the query portion of the model entity by a

"required" attribute. For example, the "patient" model entity 506_1 defines "patient id" as a required field with the provision of a required attribute 610 in the query portion 604, thereby ensuring that all query results for patients will include patient id.

[219] As an example of how the required attribute 610 is applied, consider the initial Abstract Query 700 shown in FIGURE 7. The Abstract Query 700 represents the initial form of an abstract query as specified by a user, for example. Note the explicit reference 702 to the "Patient" model entity 506_1. As a result of this reference, the logic of the DAM 502, specifically the metadata of the Patient model entity 506_1, is applied to convert the initial Abstract Query 700 into an effective Abstract Query 704. In this case, "Patient ID" was added to the result fields specified in the effective Abstract Query 704 because the "patient" model entity 506_1 defines "Patient id" as a required field with the provision of a required attribute 610.

[220] The augmentation of the effective Abstract Query 704 from an end-user's perspective is described with reference to FIGURES 8-12, which show a series of user interface screens. Referring first to FIGURE 8, a screen 800 is configured with a selection menu 802 from which a user selects a query focus. Each of the available selections corresponds to one of the defined model entities 506. Illustratively, the user selects "Patient" as the query focus and clicks the "Next" button 804, which causes the user interface to display the next screen 900 shown in FIGURE 9. That is, the user has elected to craft a query which invokes the "Patient" model entity 506_1. The user then specifies various query conditions in an input field 1002 as shown in FIGURE 10. Clicking the "Next" button 1004 causes the user interface to display the next screen 1100 shown in FIGURE 11. The screen 1100 includes a Result Fields input field 1102. Illustratively, the input field 1102 is primed with the "Patient ID" field. That is, the "Patient ID" field is automatically added to the Result Fields input field 1102 because the "Patient" model entity 506_1 defines "Patient ID" as a required field with the provision of a required attribute 610 (shown in FIGURE 6). The user may then specify additional result fields as shown in FIGURE 12. In an alternative embodiment, the "Patient ID" is not added to the result fields of the query until submitted for execution by the user.

[221] Referring now to FIGURE 13, one embodiment of a method 1300 illustrating the result field augmentation of a query is described. The augmentation process begins with receipt of an initial abstract query 1304A (step 1302), such as the initial Abstract Query 700 described with reference to FIGURE 7. An effective query 1304B (such as the effective Abstract Query 704 described with reference to FIGURE 7) is then set to the initial query 1304A (step 1306). At this point, the composition of the effective abstract query 1304B is the same as the initial abstract query 1304A. The effective abstract query is then examined for a reference to a model entity (step 1308). In the

absence of such a reference, processing is complete and the method 1300 exits. If, however, the effective abstract query 1304B includes a reference to a model entity the appropriate model entity definition 506 is retrieved from the data abstraction model 502. For each required query field (indicated by the required attribute 610) in the model entity (loop entered at step 1312), the method 1300 determines whether the required field is already specified as a result field in the initial abstract query 1304A (step 1314). If not, the required field is added to the result fields of the effective abstract query 1304B.

[222] In another aspect, model entities establish a focal point within the data environment that can be used to implement the appropriate logic needed to correlate information that spans multiple entities (e.g. tables) in the underlying physical data repository. From this focal point, a direction to interpret relationships between tables can be established. For example, the physical entity relationships specification 526 describes a 1-to-many relationship between the PatientInfo table 520 and the TestInfo table 521, since each patient can have multiple lab test results. A model entity focused on the patient entity would establish a point of reference to correlate patient information with lab test results. For example, in the case of a relational database, the model entity for "patient" would be used to determine optimal table join logic. Since each patient can have multiple lab test results, a query looking for patients with multiple test results would join the lab test table multiple times to enable selection of patients with all of the desired test results. However, a model entity focused on lab tests would only join the patient information table once since the focus is on lab tests and the relationship in the direction "lab test"-to-"patient information" is degree one (1).

[223] The effect of a model entity on query construction can be illustrated with respect to FIGURE 14. An illustrative abstract query 1400 includes a plurality of query conditions 1402, result fields 1404 and a reference 1406 to a model entity, in this case "Patient". That is, the "Patient" model entity 506_1 is specified as the focal point of the query 1400. The query conditions 1402 include two conditions with respect to a Glucose Test, where the two conditions are logically ANDed together. Since the "Patient" entity is the focus, the query conditions 1402 are interpreted to mean "find patients having both a glucose test value = 5 AND a glucose test value = 10". Further, the relationship defined in the physical entity relationships specification 526 between the "PatientInfo" table 520 and the "TestInfo" table 521 is one-to-many, indicating that a patient can have more than one test result. Given this information, it is determined that a physical query 1408 (illustratively an SQL query) corresponding to the abstract query 1400 will require two instances of the "TestInfo" table 521 in order to compare two test results for the same patient. The two instances of the "TestInfo" table 521 are identified as T2 and T3 in the selection clause of the physical query 1408. Further, the

physical entity relationships specification 526 is used to generate the necessary correlation logic 1410 between each physical entity involved in the query. In a relational model, the correlation logic is join logic specifying how tables are joined.

[224] Referring now to FIGURE 15, a physical query generation process 1500 is described for generating a physical query based on abstract query reference in a model entity. The process 1500 is initiated when a completed abstract query is received (step 1502). For each abstract query condition in the abstract query (step 1504), a series of steps is performed. Specifically, for a given abstract query condition, the process 1500 determines whether more than one ANDed condition is specified for the field of the given abstract query condition (step 1506). If not, a physical query contribution is generated against a single instance of the physical entity corresponding to the field of the given abstract query condition (step 1508). If, however, step 1506 is answered affirmatively, the physical entity corresponding to the field for the given abstract query condition is determined (step 1510). In addition, the physical entities corresponding to the specified model entity are determined (step 1512). The physical entity relationships specification 526 is then examined to determine whether a one-to-many relationship exists between any of the physical entities corresponding to the model entity and the physical entity for the field of the given abstract query condition. If not, a physical query contribution is generated against a single instance of the physical entity corresponding to the field of the given abstract query condition (step 1508). If, however, step 1514 is answered affirmatively, a physical query contribution is generated against another instance of the physical entity corresponding to the field of the given abstract query condition (step 1516).

[225] After having processed each abstract query condition, the result fields contribution for the query is generated (step 1518). Finally, correlation logic between each physical entity involved in the query is generated using the relationship metadata contained in the physical entity relationships specification 526 (step 1520). The resulting physical query can then be executed.

[226] In still another aspect, model entities 506 can be used to represent a set of query result fields. By abstracting groups of logical fields (and, hence, physical fields) applications and users are able to deal with higher level entities (e.g., a patient), without having to understand the details of what constitutes the entity.

[227] An illustration of using model entities to represent a set of query result fields is described with reference to FIGURE 16. An initial Abstract Query 1600 is shown with illustrative query conditions 1602 and result fields 1604. In this example, the initial Abstract Query 1600 includes a reference 1604 to the "Patient" model entity 506_1. As a result of this reference 1604, steps are taken to expand the result fields 1604 to include all of the logical fields defined for the "Patient" model entity 506_1, resulting in the

effective Abstract Query 1608.

[228] A result fields augmentation process 1700 is described with reference to FIGURE 17. For a given an initial abstract query 1704A (step 1702), a corresponding effective query 1704B is set (step 1706). The result fields of the query 1704A are then examined to determine a reference to a model entity (step 1708). If no such reference is identified, the process 1700 is complete. If a model entity reference is present in the query, the model entity definition 506 is retrieved (step 1710). For each required field in the model entity (step 1712), the process 1700 determines whether the required field is in the specified result fields of the initial abstract query 1704A (step 1714). If not, the required field is added to the result fields of the effective abstract query 1704B (step 1716). At the conclusion of this processing for each required field, the effective abstract query 1704B includes the full complement of logical fields for the specified model entity definition 506.

[229] It should be noted that the individual aspects separately described with reference to FIGURES 13, 15 and 17 (and related figures) may be used in combination. For example, assume that the initial Abstract Query 1400 shown in FIGURE 14 does not include the "Patient id" field in the result fields 1404. A first stage of processing may be performed according to the method 1300 whereby the "Patient id" field is added to the result fields 1404. A second stage of processing is then performed according to the process 1500 to generate the physical query 1408. Persons skilled in the art will recognize other process combinations which may be performed.

[230] Modification Operations Using Model Entities

[231] Aspects of the invention are described above with reference to accessing data for the purpose of returning results. In the case of SQL, these are SELECTION operations. However, modification operations are also supported, including well-known modification operations such as INSERT, DELETE and UPDATE and the like. Accordingly, the following describes embodiments extending and enhancing the functionality of the abstract framework described above to support modification operations using model entities.

[232] Since a model entity may span multiple physical entities (e.g., tables), multiple database operations may be needed to implement a modification operation. That is, embodiments are provided for modifying physical data via a single logical operation spanning multiple statements (e.g., multiple SQL statements) issued against the physical data. To this end, model entities define a minimal set of fields used to derive the complete set of fields involved in data insert and delete operations for an instance of the model entity. For example, patient information spans two tables (e.g., the "PatientInfo" table 520 and the "AddressInfo" table 521) in the relational data source 504 implementation shown in FIGURE 5. To implement inserts and deletes, the model

entity for patient identifies at least one field in each table to serve as a "seed" in determining the complete set of fields that are needed to insert a new patient into the database, as well as the complete set of tables that are involved to delete a patient from the database. Specifically, the seed fields are specified in insert portion 606_1 and delete portions 608_1 of the model entity definition 506_1 for "Patient" in the DAM 502 shown in FIGURE 6. In the case of the insert portion 606_1, the seed field corresponding to the "PatientInfo" table 520 is "Last Name" and the seed field corresponding to the "AddressInfo" table 521 is "Street". Each model entity 506 defined in the DAM 502 may have a similar portions specifying seed fields.

[233] Based on the seed fields, multiple physical operations are performed against the data repository to implement a single abstract operation. For inserting into a relational data source 504, for example, this would involve creation of multiple physical SQL INSERT statements for the tables involved. The application/user need only specify a model entity, which is then used to identify the corresponding physical entities and related logical fields involved in the modification operation. The model entity may be selected from a drop-down menu of an HTML form, for example. Further, deletes allow for conditions to be specified that can be used to target the changes required in the physical data repository.

[234] FIGURE 18 shows a simplified view of the environment 500 and includes an illustrative abstract insert operation specification 1802_1 and a delete operation specification 1802_2 (collectively, abstract operation specifications 1802) used to implement an insert operation and a delete operation, respectively, against the relational data source 504. The abstract operation specifications 1802 are composed via the data abstraction model 502 according to specifications provided by a requesting entity 512 (e.g., a user/application). In each case, the abstract operation specifications 1802 specify a seed value for each affected physical entity. The seed values are retrieved from the appropriate portion of the model entity 506 selected by the requesting entity 512. Based on the seed values, related logical fields are determined and made a part of the abstract operation specifications 1802. Thus, as in the case of queries, the framework of the present application will provide the requesting entity 512 with the related fields according to the specified model entity 506, rather than requiring that knowledge about a database schema at the application level or end user. Values may then be supplied for each of the fields, either from the requesting entity 512 (e.g., a user) or from some other source such as the value generator 524.

[235] Although in the present examples, each abstract operation includes two seed fields, an abstract operation may also be implemented with only a single seed field. For example, the seed field may be a primary key having an associated foreign key. In this case, an abstract operation specifying the seed field may affect the physical data on

which the foreign key is defined. It should be noted that in some cases this result may not be desirable. That is, it may be undesirable to propagate changes based on primary key/foreign key relationships. If propagation is desired, only one statement directed to modifying the table containing the primary key is needed. The DBMS will handle modifications to the related tables. In some instances the DBMS may not support propagation, in which case multiple statements are needed. Although not shown the data abstraction model may include an attribute specifying whether propagation is desired or not for a given logical field.

[236] In addition to seed fields, the DAM defines other field types used to implement modification operations. Generally, such fields may be required or optional. "Required" means that the requesting entity must supply a value for the field because there is no suitable default value, no algorithm to generate a value and the field cannot be null. Required fields are defined by the required attribute 514, shown in FIGURE 5. An optional field is one which does not require specification of a value by the requesting entity. Optional fields include: 1) fields that can be assigned the value of NULL; 2) fields that have an algorithm that can be used to generate a value for that field (referred to herein as "generated fields"); and 3) fields that have a defined default value in the DAM (referred to herein as "default fields"). Generated field values are generated by a value generator 524 (i.e., an algorithm). Generated fields are defined by the generated attribute 516, shown in FIGURE 5. Default values are used where no name/value pair was specified for a particular field related to the entity defined by a seed field. Default fields are defined by the default attribute 518. Default values may be statically defined or generated. As an example of a generated default value, the Test Date value in the illustrative abstract delete specification 506_3 defaults to the current date. The requesting entity (e.g., user) may be given the option of supplying a different value.

[237] As an example of required and optional fields, consider the logical fields corresponding to the patient entity. It was noted above that the patient entity is logically defined by the "Patient ID" logical field specification 510_1, the "First Name" logical field specification 510_2 and the "Last Name" logical field specification 510_3. The "First Name" logical field specification 510_2 and the "Last Name" logical field specification 510_3 include the required attribute and are required fields. In contrast, the other logical fields defining the patient entity (i.e., birth date and gender) are optional.

[238] Accordingly, inserts and updates to the data source 504 provide for identification of the actual fields (i.e. columns) that are to be modified along with the new value to be put in the data source. A set of name/value pairs represents the fields/values within the data abstraction model 502 that correspond to the physical fields/values to be modified. The name represents the abstract name for the logical field that is mapped via the data

abstraction model 502 to its underlying physical representation. The value is the abstract value to be inserted or updated in the data source for that field. In one aspect, using an abstract data model (i.e., the data abstraction model 502) allows for automatic conversion of abstract values to the correct physical values to be inserted into the database. For example, the data abstraction model 502 can be defined to use values such as "Male" and "Female" for gender when the underlying physical data repository may use values of "F" and "M" to represent those values, as illustrated by the patient information table 520, which contains a record having the value "F" in the gender (end) column. Input values for an insert operation take advantage of those abstract value specifications to provide further cushion for changes to the underlying physical representation of not only the structure of the underlying data repository, but also from the physical form of the data contained within the data repository.

[239] Some situations require special considerations. In many cases, a logical field may be physically represented in multiple fields in a physical data repository. One example would be the case in the relational data source 504 where a column in one table was defined as a foreign key to a column in another database table. For example, a patient ID may be a primary key within the patient information table 520 and may also be defined as a foreign key within the test information table 522. Although this is a typical example, an abstract relationship between two fields does not necessarily have to be enforced by a physical relationship (such as primary/foreign key) in the underlying data repository. Using metadata about the relationship between the two physical locations (i.e., relationship between column(s) in a first table to column(s) in a second table), a single field within the abstract data representation can be used to handle both physical locations for the field. The application is shielded from the knowledge of the various places a particular value is used within the database schema.

[240] Special considerations must be taken into account when these related fields are included on an insert or delete operation. When performing an insert operation, the runtime/DAM logic must recognize the various physical representations for the single abstract data field. The correct physical representation must be used based on the focus item of the insert operation. For example, if the patient identifier was represented as column "ID" in the patient information table 520 and as column "PID" in the test information table 522, the correct column name must be identified based on the table defined for the insert operation by the focus item. Additionally, these additional forms of the physical representation must be taken into account when determining default values and required values for the insert operation. For example, if the focus item identified that the underlying physical table for the operation was the test information table 522, the runtime/DAM logic must recognize that the abstract patient ID field ("PID") must be considered when looking for default and required values. That is, a

new patient ID cannot be generated.

[241] Additional considerations must be given to delete operations when dealing with abstract fields that represent multiple locations in the physical data repository to ensure that data integrity is maintained. The underlying data repository may enforce additional restrictions on these types of fields. Relational databases provide aspects such as restricting updates or deletes to columns defined with a primary key/foreign key relationship or cascading those updates and deletes through the foreign key tables. That is, a delete against a primary table such as the patient information table 520 could be set up to indicate that the delete should cascade and delete the corresponding rows from the test information table 522 based on the primary key/foreign key relationship based on patient ID. Using an abstract representation of the data repository, the implementation can choose whether to restrict these operations or attempt to propagate the changes through the various physical entities based on the definition of the relationships for the abstract field.

[242] Since a modification based on specification of a model entity may involve multiple physical operations, the sequence in which the operations should be performed must also be considered. In the current data mining applications based on SQL, for example, the application is required to have the knowledge of order dependencies between operations. An aspect of the present invention decoupages this knowledge from the application. In one embodiment, order dependencies between operations are specified in the physical entity relationships specification 526.

[243] Referring now to FIGURE 19, a method 1900 illustrates the interaction between requesting entity 512 and the data abstraction model 502 in the case of composing an abstract insert specification. For purposes of illustration it will be assumed that the requesting entity 512 is representative of the application 140 (FIGURE 1), which receives input from a user via a user interface (e.g. the browser program 122 FIGURE 1). Initially, the user selects a model entity (step 1902). The seed fields for the selected model entity are then determined and the abstract insert specification 1802_1 is updated with the seed field (step 1906). Once each of the seed fields has been determined, the requesting entity 512 issues a request for the required and optional fields according to the specified seed fields (step 1908). The data abstraction model 502 is invoked to determine required and optional fields for the insert operation (step 1910). An illustrative representation of the processing occurring at step 1910 is described below with reference to FIGURES 20A-B. Having made the determination at step 1910 the abstract insert specification 1802_1 is initialized with the required and optional fields (step 1912). The required and optional fields are then returned to the requesting entity (step 1914), which prompts the user to provide values for each of the fields (step 1916). Well-known techniques in the art of user interfaces may be used to identify and

distinguish for the user required fields and optional fields. For example, required fields may be highlighted in red, marked with an asterisk, or include a parenthetical comment indicating that the field is required. In an alternative embodiment, the application 140 itself may provide all or some of the values. Once values for at least each of the required fields (and any optional fields) has been specified (step 1918), the abstract insert specification 1802_1 is populated with the specified values (step 1920).

[244] Referring now to FIGURES 20A-B, one embodiment of step 1910 for determining required and optional fields is shown. After accessing the abstract insert specification 1802_1 to retrieve the seed fields (step 2002) specified by the requesting entity 512, the appropriate logical field specification of the data abstraction model 502 is referred to in order to determine the physical entities (e.g., tables in the relational data source 504 shown in FIGURE 5) corresponding to the seed fields (step 2004). In the case of an insert operation, for each identified physical entity, the data repository abstraction is used to determine other logical fields associated with the same physical entity (steps 2006 and 2008). A loop is then entered (at step 2010) for each of the determined related logical fields that define a particular physical entity referenced by a seed field. That is, a series of steps is performed for each of the related logical fields of each physical entity. For a given logical field of a given physical entity, a determination is made as to whether a key relationship for the given logical field has already been processed (step 2012). For the first iteration of the loop entered at step 2006 the determination made at step 2012 is answered in the negative. During subsequent iterations, step 2012 ensures that once a value has been specified for a field, a subsequent and conflicting value will not be specified. Processing is then performed to determine whether the field is a required field (at step 2020), whether the field is a default value field (step 2026), or whether the field is a generated value field (step 2034). The field type is determined according to the attribute (i.e., the required attribute 514, the generate attribute 516 or the default attribute 518) present in the logical field specification for the current field being processed by the loop. If the field is required (step 2020), the field is added (step 2022) to a required field list 2024. In the case of a default attribute (step 2026), the field value is initialized with a default value (step 2028). Where the logical field specification includes a generate attribute 518 (step 2034), the field is initialized with a generated value (step 2036). In the case of both generated values and default values, the corresponding fields are added (step 2030) to an optional fields list 2032. If the field is not defined as any one of required, default or generated, then the field is initialized with a NULL value (step 2038) and then added (step 2030) to the optional field list 2032.

[245] Returning to step 2012, if the current field being processed is in a key relationship (e.g., primary key/foreign key relationship) with another field which has already been

processed, then the value for the current field is set to the value of the previously processed related field (step 2014). The current field is then added to an implicit field list 2018 (step 2016). Accordingly, implicit fields are created with multiple physical entities are involved in the abstract operation and those entities have key relationships. In this case, only one field and one value is exposed through the interface. The other field in the pair of key fields is considered implicit; it does not have to be specified as part of the abstract insert and will take on the same value as the corresponding key in the pair. The implicit fields are not exposed to the requesting entity, but are accounted for when the abstract insert is converted into a concrete (i.e., executable) insert statement, as will be described below with reference to FIGURE 21.

[246] Once each identified related field is processed according to the loop entered at step 2010, the processing is repeated for the next entity (step 2006). Once each entity has been processed, the processing to determine required and optional fields is complete. Accordingly, the abstract insert specification 1802_1 is updated according to the required fields list 2014 and optional fields list 2022 (step 1912), and the required fields and optional fields are then provided to the requesting entity 512 (step 1914), as shown in FIGURE 19.

[247] Having composed the abstract insert specification 1802_1, the insert operation may be executed. FIGURE 21 shows one embodiment of a method 2100 for executing the insert operation according to the abstract insert specification 1802_1. Generally, upon submission of a request to execute the insert from the requesting entity 512, the run-time component 150 (described above with reference to FIGURE 1) is invoked convert the abstract insert specification 1802_1 to a physical insert operation. The physical insert operation is then executed.

[248] Conversion of the abstract insert specification 1802_1 to a physical insert operation is initiated by grouping fields (from the implicit field list 2018, the required field list 2024, and the optional field list 2032) according to their respective physical entities (2104). In particular, the run-time component 150 then enters a loop (step 2106) for each physical entity and a sub-loop (step 2108) for each logical field of a given physical entity. For a given logical field, the physical location of the field is determined from the data abstraction model 502 (step 2110). A physical location list 2114 is then updated with the determined physical location (step 2112). In some cases, the logical field may have an internal value (determined at step 2116). That is, the value of the logical field may be different from the value for the physical field. For example, the logical field name may be "Male" while the physical field name is "M". In this case, the value must be made consistent with physical value. This is done by updating a value list 2122 with the internal value (step 2118). If the field values are not different (i.e., step 2116 is answered negatively), the value list 2122 is updated with the

given value for the physical field (step 2120).

[249] Once the processing for the loop entered at step 2108 has been performed for each logical field in the abstract insert specification 1802_1, for a given physical entity, a physical insert statement is built from the location list 2114 and the value list 2122 (step 2124). The physical insert statement is then added to an insert statement list 2122 (step 2126). The foregoing processing is then repeated successively for each entity (step 2106). Subsequently, an ordering algorithm is performed on the insert statement list 2128 (step 2130). One embodiment of the ordering algorithms is described with reference to FIGURE 22. The physical insert operation is then executed (step 2132).

[250] Referring now to FIGURE 22, an embodiment of the ordering algorithm performed at step 2130 is described. Initially, a "sorted flag" is set to False (step 2202). A series of steps are then performed for each insert statement in the insert statement list 2128 until the "sorted flag" is set to True (steps 2206, 2208 and 2210). Specifically, for a given insert statement in the insert statement list 2128 (beginning with the first insert statement in the list), the corresponding entity is determined (step 2212). Then, the relationship between the corresponding entity of the given insert statement and each related entity of the remaining insert statements in the insert statement list 2128 is determined (step 2214 and 2216). Specifically, the run-time component 150 determines (with respect to the physical entity relationships specification 526) whether the entity of the given insert statement is a secondary entity with respect to a primary related entity of another insert statement (step 2216). If so, the given insert statement is moved to a position after the insert statement of the related entity (step 2218). This process is repeated until the insert statement list 2128 can be traversed without encountering a current entity which is secondary with respect to an entity of a subsequent statement in the insert statement list 2128. At this point, the physical insert statements in the insert statement list 2128 are ordered according to the hierarchical relationship specified in the physical entity relationships specification 526. This process ensures that a primary entity containing a primary key of a primary/foreign key pair is inserted into first, before the secondary entity (containing the foreign key) is modified.

[251] As noted above, aspects of the invention may "guide" the requesting entity 512 (i.e., the application 140) through the process of building an abstract modification operation. This aspect can be illustrated for an insert operation with reference to FIGURES 23-26 where HTML forms 160 (shown in FIGURE 1) are intelligently populated to indicate characteristics of various fields. Referring first FIGURE 23, a user interface screen 2300 is shown which may be displayed when a user elects create an abstract insert. The screen 2300 includes a menu 2302 of available entities which may be inserted into. The entity selected by the user from the menu 2302 are displayed in a "Selected Entity" field 2304, as shown in FIGURE 24. In this example, the user has selected

"Patient". Upon making the desired selections, the user clicks the "Next" button 2306 to submit the selections and proceed to the next screen 2500 shown in FIGURE 25. The screen 2500 is formatted with a plurality of input fields 2502 AI which are selected according to the entity specified by the user in the "Selected Entity" field 2304. That is, the fields defined for the selected entity (i.e., "Last Name" and "State" in this example) are used as seed fields in determining which fields to display in the screen 2500. In this case, "Last Name" is a logical field 510_3 corresponding to the "PatientInfo" table 520 for which the "First Name" logical field 510_2, "Birth Date" logical field 510_4 and "Gender" logical field 510_6 are also specified as related logical fields. As such, each of these logical fields is displayed as an input field (input fields 2502C, 2502E and 2502F, respectively) in the screen 2500. Similarly, the seed field "State" is used to identify the corresponding physical entity (i.e., AddressInfo table 521) and its related logical fields. Further, required fields (as defined by the presence of a required attribute 514 in the DAM 502) are marked with an asterisk (*). In this case, the input fields 2502A, 2502B and 2502C are marked as required fields. Further, generated in default fields are primed with the appropriate values. In the present example, the "Patient ID" input field 2502D is primed with a generated value. A default values not shown because none of the input fields 2502 correspond logical field specifications having a default attribute 518.

[252] Generally, the method 2700 describes the interaction between the requesting entity 512 and the data repository abstraction 502, which implements the abstract delete. As in each of the previous abstract modification operations, the abstract delete requires composing an abstract specification. To this end, the requesting entity 512 specifies a model entity to delete (step 2702). The seed fields for the selected model entity are then determined (step 2704) which the data abstraction model 502 uses to create/update abstract delete logic of the abstract delete specification 1802_2 (step 2706). The requesting entity 512 then provides selection conditions for selection of data to delete (step 2708). The selection conditions are added to the selection portion of the abstract delete specification 1802_2 (step 2710).

[253] The abstract delete specification is then used by the run-time component 150 to generate an executable physical delete specification. One conversion method 2800 for converting the abstract delete to a physical delete, is described with reference to FIGURE 28. The conversion process is initiated when the requesting entity 512 submits a request to execute the delete operation. The run-time component 150 first groups the specified seed fields according to their respective physical entity (step 2802). That is, the run-time component 150 uses the seed value (specified by the requesting entity 512 at step 2702) and the data abstraction model 502 to locate the physical entity to delete from. In particular, the logical field specification of the data

abstraction model 502 corresponding to the seed field is identified. The identified logical field specification provides the necessary logic (i.e., the appropriate access method) to access the physical entity to delete from. For each physical entity (step 2804), the run-time component 150 generates selection logic according to the selection conditions specified in the abstract delete specification 2402_2 (step 2806). Using the determined physical entity and the generated selection logic, the run-time component 150 builds a physical delete statement (step 2808) which is added to a delete statement list 2812 (step 2810). The statements in the delete statement list 2812 are then ordered (step 2814) and executed (step 2816).

[254] One embodiment for generating the selection logic at step 2806 of the method 2800 is shown in FIGURE 29. It is noted that step 2806 substantially involves performing steps 306, 308, 310 and 312 of FIGURE 3. Accordingly, for each selection criterion (step 2902), the specified field definition is retrieved from the abstraction component 502 (step 2904). A concrete/physical selection contribution is built (step 2906) and then added to the selection portion of the update statement (step 2908). The logic for building the concrete/physical selection contribution is substantially the same as was described for queries with respect to FIGURE 4 and, therefore, will not be described again in detail here.

[255] One embodiment of the ordering performed at step 2814 is described with reference to FIGURE 30. Initially, a "sorted flag" is set to False (step 3002). A series of steps are then performed for each delete statement in the delete statement list 2812 until the "sorted flag" is set to True (steps 3006, 3008 and 3010). Specifically, for a given delete statement in the delete statement list 2812 (beginning with the first delete statement in the list), the corresponding entity is determined (step 3012). Then, the relationship between the corresponding entity of the given insert statement and each related entity of the remaining delete statements in the delete statement list 2812 is determined (step 3014 and 3016). Specifically, the run-time component 150 determines (with respect to the physical entity relationships specification 526) whether the entity of the given delete statement is a primary entity with respect to a secondary related entity of another delete statement (step 3016). If so, the given delete statement is moved to a position after the delete statement of the related entity (step 3018). This process is repeated until the delete statement list 2812 can be traversed without encountering a current entity which is primary with respect to an entity of a subsequent statement in the delete statement list 2812. At this point, the physical delete statements in the delete statement list 2812 are ordered according to the interrelationship specified in the physical entity relationships specification 526. This process ensures that a primary entity containing a primary key of a primary/foreign key pair is deleted last, after the secondary entity (containing the foreign key) is deleted.

[256] It should be noted that the embodiments described above are merely illustrative and not exclusive. Persons skilled in the art will recognize other embodiments within the scope of the invention. For example, the foregoing describes an embodiment in which order dependencies are defined as part of the DAM 148 (i.e., order dependencies are defined in the physical entity relationships specification 526). Thus, changes in order dependencies require changes in the abstract data representation, but allow the application to be used without changes. An alternative embodiment provides for a less rigid definition of order dependencies by allowing for a higher degree of automation in the determination dependencies. That is, given the physical entity relationships specification 526 defined in an abstract data representation, the sequencing of operations could be determined dynamically by applying a rule set such as the following: (i) insert operations involving a key field need to insert into the primary entity before any related entities; (ii) update operations involving a key field could be restricted or automatically propagated from the primary entity first followed by all related entities; (iii) delete operations involving a row that includes a key field could be restricted or automatically propagated from the secondary entities to the primary entity; and (iv) operations for completely unrelated entities would be executed in any order.

[257] In still another embodiment, it is contemplated that the database is checked for referential integrity cascade operations. In this case, the database itself handles some of the work and allows DAM 148 to effectively 'ignore' those low level operations, and only perform the high level ones. As such, this invention can 'patch' referential integrity holes in legacy databases that may no longer be fixable at the database layer because of assumptions built into legacy applications that use the database.

[258] FEE SCHEDULES

[259] In one embodiment, a data abstraction model is configured with fee schedules for one or more logical field specifications, individually or collectively. In general, a fee schedule indicates a cost of involving a particular logical field(s) in an abstract operation, such as a query, insert, delete or update. It is contemplated that a fee structure may be imposed on a single logical field specification, a category of logical fields and/or on model entities. In each case, a corresponding fee structure is implemented as metadata of the data abstraction model. For example, referring to FIGURE 31 a simplified representation of FIGURE 5 is shown, in which the logical field specification 510_{12} for Glucose Test has an associated fee schedule 530A. For simplicity, only one logical field specification 510 is shown having a fee schedule; however, any number of logical field specifications of the DAM 502 may include fee schedules. A category configured with a fee schedule 530B is represented by the Demographic category 508_1. Again, only one category is shown having a fee schedule, for simplicity. Finally, each of the model entities $506_{1\text{-}3}$ is shown with an associated fee

schedule 530C-E, respectively.

[260] Embodiments of the fee schedules 530A-E can be seen in FIGURE 32 which shows a representative detailed view of the DAM 502 shown in FIGURE 31. For example, the fee schedule 530B of the Demographic category 508_1 includes individually defined fees for queries, inserts and updates. For each operation the fee may be per request (as illustrated by the query fee and the update fee), or per item (as illustrated by the insert fee). Similarly, the Glucose Test field specification 510_{12} includes a per item query fee schedule 530A (which in this case is one attribute).

[261] Referring now to FIGURE 33, embodiment of the model entity specifications 602_{1-2} is shown. Other elements of the model entities specifications 602_{1-2} were described above with respect to FIGURE 6, and are not repeated here. In the case of model entities, the attributes of the respective fee schedules may be distributed over the portions for each operation. Thus, the "Patient" model entity 506_1 includes a per request query fee attribute in the query portion 604_1 of the entity and a per item insert fee attribute in the insert portion 604_2 of the entity, and the "Test" model entity 506_2 includes a per item query fee attribute in the query portion 604_2 of the entity. For brevity, the "Account" model entity 506_3 is not shown.

[262] The embodiments described with respect to FIGURES 31-33 illustrate that fees may be based on various levels of granularity of data. These embodiments are merely illustrative and persons skilled in the art will recognize other embodiments within the scope of the invention. For example, fees could be made to vary with volume such that (i) queries returning high volumes of results incur a penalty (which may be desirable to deter users from draining the performance resources of a system); (ii) or, queries returning high volumes of results receive a discount (which may be useful for a data provider to establish a competitive advantage and/or increase profits by encouraging increased use).

[263] Embodiments of fee calculation algorithms for various abstract operations are shown in FIGURES 34-35. For convenience and simplicity, the fee calculations for queries and inserts are described together with respect to FIGURE 34. Fee calculations for updates are described separately with respect FIGURE 35. In each case, the fee calculation is implemented by the fee calculator 151 shown in FIGURE 1. Note that at least some of the operations of the fee calculator 151 could be performed as part of the abstract operation runtime, or could be implemented as an "after the fact" process, using transaction and audit logs to compute the applicable charges sometime after execution of the transaction. In this case, it is contemplated that the user may be warned (by operation of the fee calculator 151) that fees apply to a given transaction, prior to executing the transaction.

[264] Referring now to FIGURE 34, one embodiment of a fee calculation algorithm 3400

for queries and inserts is shown. The fee calculation algorithm 3400 is initiated (at step 3402) for a given abstract operation. The fee calculator 151 then determines whether the abstract operation includes a reference to a model entity (step 3404). If so, the fee calculator 151 determines (at step 3406) whether the reference model entity defines a fee for the particular operation (i.e., a query fee or an insert fee). If so, the fee calculator 151 determines (at step 3408) whether the fee is a per item fee. If so, a "per item fee list" 3412 is updated (at step 3410). Otherwise, the fee is assumed to be a per request fee, in which case a "per request fee list" 3416 is updated (step 3414). In one embodiment, the per item fee list 3412 includes a reference to each item for which a fee applies, as well as the corresponding fee. In contrast, the per request fee list 3416 need only include a record of the fee for each request made with respect to the given field. In either case, the algorithm 3400 then enters a loop (at step 3418) for each field specified in the given abstract operation. The loop is also entered from steps 3404 and 3406 if either step is answered in the negative.

[265] For a given field of the abstract operation, the algorithm 3400 determines (at step 3420) whether an operation fee (i.e., query fee or insert fee) is defined for the field. If not, processing proceeds to step 3428 described below. Otherwise, the fee calculator 151 determines (at step 3422) whether the operation fee is per item. If so, the fee calculator 151 updates the per item fee list 3412 (at step 3426). Otherwise, the per request fee list 3416 is updated (at step 3424).

[266] The fee calculator 151 then identifies (at step 3428) the parent category for the given field being processed. A loop is then entered (at step 3430) for each category containing the given field being processed (i.e., the parent category, grandparent, great grandparent, etc.). For a given category, the fee calculator 151 determines (at step 3432) whether an operation fee is defined for the category. For any given category having a defined operation fee, the fee calculator 151 then determines (at step 3434) whether the fee is a per item fee. If so, the per item fee list 3412 is updated (step 3436); otherwise, the per request fee list 3416 is updated (step 3438). After each category for a given parent category has been processed, the algorithm 3400 returns to step 3418 to begin processing the next field in the abstract operation. Once each field of the abstract operation has been processed in the foregoing manner, the total fee for the abstract operation can be calculated based on the information contained in the per item fee list 3412 and the per request fee list 3416.

[267] In one embodiment, the fee calculator 151 first initializes a total fee to zero (at step 3440). The operation is then executed (at step 3442). In this regard it is contemplated that the fee calculator 151 is not responsible for executing the operation, and step 3442 is shown merely for convenience of explanation. Before, during or after execution, the fee calculator 151 sums (at step 3444) the per request fees (based on the information in

the per request fee list 3416), and adds (at step 3446) the sum to the total fee. The fee calculator 151 then determines (step 3444) the number (N) of query results (in the case where the operation is a query), or the number of items (in the case where the operation is an insert). For each per item fee (F), the fee calculator 151 calculates the product of N and F, and adds the product to the total fee (steps 3450 and 3452).

[268] Referring now to FIGURE 35, one embodiment of a fee calculation algorithm 3500 (performed by the fee calculator 151) for update operations is described. The algorithm 3500 is initiated for a given abstract update operation (at step 3502). The fee check letter 151 then enters a loop (step 3504) for each field in update. For a given field, the fee calculator 151 determines (step 3506) whether the update fee is defined for the field. If so, the fee calculator 151 determines (step 3508) whether the update fee is per item. If so, the per item fee list 3412 is updated (at step 3512). Otherwise, the fee calculator 151 updates the per request fee list 3416 (at step 3560).

[269] The fee calculator 151 then identifies (at step 3514) the parent category for the given field being processed. A loop is then entered (at step 3516) for each category containing the given field being processed (i.e., the parent category and ancestor category for that parent). For a given category, the fee calculator 151 determines (at step 3516) whether an update fee is defined for the category. For any given category having a defined update fee, the fee calculator 151 then determines (at step 3520) whether the fee is a per item fee. If so, the per item fee list 3412 is updated (step 3522); otherwise, the per request fee list 3416 is updated (step 3524). After each category for a given parent category has been processed, the algorithm 3500 returns to step 3516 to begin processing the next field in the abstract update operation.

[270] Once each field of the abstract update operation has been processed in the foregoing manner, the fee calculator 151 determines (at step 3526) whether the given field is referenced in a model entity. If so, the fee calculator 151 determines (at step 3528) whether the reference model entity defines a fee for update operations. If so, the fee calculator 151 determines (at step 3530) whether the fee is a per item fee. If so, the per item fee list 3412 is updated (at step 3532). Otherwise, the fee is a per request fee, in which case a per request fee list 3416 is updated (step 3534).

[271] Once each field involved in the update operation has been processed, the total fee for the abstract operation can be calculated based on the information contained in the per item fee list 3412 and the per request fee list 3416. In one embodiment, the fee calculator 151 first initializes a total fee to zero (at step 3540). The operation is then executed (at step 3542). In this regard it is contemplated that the fee calculator 151 is not responsible for executing the operation, and step 3542 is shown merely for convenience of explanation. Before, during or after execution, the fee calculator 151 sums (at step 3544) the per request fees (based on the information in the per request fee

list 3516), and adds (at step 3546) the sum to the total fee. The fee calculator 151 then determines (step 3544) the number (N) of items updated. For each per item fee (F), the fee calculator 151 calculates the product of N and F, and adds the product to the total fee (steps 3550 and 3552).

[272] One embodiment for applying a fee-based model is illustrated with reference to FIGURES 36-40, which show user interface screens (e.g., of the browser program 122 shown in FIGURE 1) for creating an abstract query. Persons skilled in the art will recognize that similar screens may be used for INSERT and UPDATE operations.

[273] Referring first to FIGURE 36, a user interface screen 3600 is shown having a model entity selection list 3602. A representation is provided in the list 3602 for each of the three model entities defined by the DAM 502 shown in FIGURE 31 (i.e., Account, Patient and Test). The screen 3600 also includes any "Per Request Fee" field 3604 and a "Per Item Fee" field 3606. The fields 3604, 3606 display a running total of fees generated for the abstract query being created by the user. Accordingly, the fields 3604, 3606 are initially empty, or indicate a zero dollar amount. A user then selects one of the model entities from the list 3602 and clicks the Next button 3608, whereby the user is presented with the user interface screen 3700 shown in FIGURE 37. For purposes of illustration, it is assumed that the user selects the Patient model entity from the list 3602. Accordingly, the "Per Request Fee" field 3604 now displays a one dollar ($1.00) fee. The one dollar fee is calculated with reference to the model entity specification 525 shown in FIGURE 33, wherein the query portion 604_1 of the Patient model entity 506_1 defines a one dollar fee, per request. The screen 3700 also includes an input field 3702 in which the user specifies the desired query conditions by clicking on the ADD button 3707, which invokes a condition selection menu (not shown). The user then clicks the Next button 3608. Illustrative query conditions are shown in FIGURE 38. In the present example the user's query includes the logical field Glucose Test, which has been associated fee of $0.50 per item (as shown by the fee schedule 530A of FIGURE 32). Accordingly, the "Per Item Fee" field of 3606 displays the $0.50 fee. If the query includes other logical fields having associated per item fees, then additional "Per Item Fee" fields may be provided in the interface 3700. Upon specifying the desired query conditions the user clicks the Next button 3608 and is then presented with the user interface screen 3900 shown in FIGURE 39. The user interface screen 3900 includes an input field 3902 for displaying the result fields to be returned by the query. The input field 3902 displays any required fields defined for the selected model entity, which for the selected Patient model entity includes "Patient ID". The input field 3902 also displays any user-selected result fields, which, in one embodiment, are added by clicking an ADD button 3904 to invoke a result fields selection menu (not shown). In this example, the user-specified result fields include

Gender, Glucose Test and Age as shown in FIGURE 40. For each result fields specified, the fee calculator 151 determines whether additional fees apply. In the present example, no additional fees apply for the specified result fields. Once the desired abstract query is completed, the user clicks the Finish button 3616, whereby the query is executed and the results are returned. An illustrative results screen 4100 is shown in FIGURE 41. A "Total Fee" field 4102 displays the total calculated fee for the query, in this example $4.00. The $4.00 is the sum of the $2.00 per request fee and a $2.00 for 4 results involving the Glucose Test field at $0.50 each (4*$0.50 = $2.00).

[274] Once the total fee for an operation is calculated, the fee may be displayed to the user. A variety of payment models may be applied to collect payment from users. For example, in one embodiment the user may be a subscriber with an account maintained by a data provider. After each transaction, or after the user logs off, the total fee may be automatically charged to the user's account. In another embodiment, the operations may be performed on computers under the control of a business entity i.e., computers at a store which provides network access for users. In this case, users may be charged prior to leaving the store.

[275] Accordingly, embodiments described herein provide a metadata-based model for establishing a usage fee for individual data items or collections of data, whereby the fees associated for one set of data can differ from the fees established for other information within the same data warehouse or federated data environment. In addition to variation in fee based on the information accessed, the proposed fee model also takes into account the type of operation performed (e.g., data retrieval, data insertion or data update). This model allows for a different fee schedule for storage of new information than the schedule used for retrieval of existing information. A fee calculator takes into account the transactions performed by a requesting entity and the fee schedules in place for information accessed to arrive at a net charge for each transaction against the data repository. In one embodiment, this approach allows for both high value and low value information to be managed within the same IT infrastructure, using the same data update and access applications for each while establishing different charges based on the value of information accessed and or level of services provided for information stored.

[276] In various embodiments, numerous advantages may be provided. In some cases, these advantages may be substantial improvements over the prior art. Some of these advantages are now described (and others have been mentioned above) with reference to specific IT contexts. However, whether or not an embodiment achieves an advantage, and whether or not such an advantage may be considered a substantial improvement, is not limiting of the invention. Therefore, the advantages and contexts described below do not define or limit the invention, which is limited only by the

claims that follow.

[277] In one embodiment a medical research database contains demographic information and associated gene expression data for each subject. The database manager would like to charge a premium for use of gene expression data which involves a more time consuming and expensive process to derive. The logical fields associated with gene expression results could have specific fees established that would allow for recovery of cost (and profit) driven by the value of this information to other parties.

[278] In another embodiment a medical records service provider who manages storage of medical record information for multiple institutions, would like to charge a premium for storage of information on new patients since the addition of new patients requires a flow of activity to establish the level of access the patient wishes to provide to their personal information. The ability to establish a specific fee structure based on the "Patient" model entity and, more specifically, on insert operations performed for this entity allows distinction in the fees for this type of requests versus other requests that have fewer associated services provided (like updating of the patient's street address).

[279] In another embodiment, a service provider would like to establish an access fee schedule for a collection of fields that provide similar value. The ability to establish fees based on a logical group or category of fields makes this possible.

[280] Some of the foregoing embodiments describe fee-based data access with reference to a data abstraction model. However, embodiments of the invention are not limited to fee arrangements implemented in a data abstraction model. Instead, any fee arrangement which provides for fees based on the particular information being accessed and/or the nature of the operation to be performed on the information (query, insert and update) is contemplated. Thus, for example, the fee metadata of the above-described data abstraction models may also represent fee schedules for physical fields (e.g., the physical fields of the database 504 shown in FIGURE 5), directly (i.e., not logically described by an abstraction model). For example, a fee table in a database is contemplated in which the fee table defines a fee schedule for requests to access or modify entities (tables) or items within an entity (column within a table). This fee schedule could also be defined to be sensitive to the type of operation performed and could indicate whether the fee was a per request or per item fee. For example, the following table could be used as a fee schedule based on a physical, relational model. Considering the Test Entity for purposes of illustration, this fee schedule would result in a $0.50 per item fee for information returned from the test table and an additional $0.50 per item fee for queries returning data from the glucose column.

Table 1

Entity	Item	Operation	Per Request Fee	Per Item Fee

DemoGraphic		Query	$1.00	$0.00
DemoGraphic		Insert	$0.00	$2.50
DemoGraphic		Update	$0.00	$0.50
Test		Query	$0.00	$0.50
Test	Glucose	Query	$0.00	$0.50

[281] While the foregoing is directed to embodiments of the present invention, other and further embodiments of the invention may be devised without departing from the basic scope thereof, and the scope thereof is determined by the claims that follow.

Claims

[001] 1. A method of providing fee-based access to data, comprising: roviding an abstract model for logically defining abstract operations to access the data, the abstract model comprising: (i) a plurality of logical fields; (ii) a mapping rule for each of the plurality of logical fields, which map the plurality of logical fields to physical entities of the data; and (iii) a fee schedule for each of the plurality of logical fields, wherein each fee schedule for a given logical field defines a fee to be charged when the given logical field is involved in an abstract operation to access a physical entity corresponding to the given logical field.

[002] The method of claim 1, further comprising: accessing the data according to an abstract operation comprising at least two of the plurality of logical fields; and calculating the fee to be charged based on separate fee schedules corresponding to each of the at least two plurality of logical fields.

[003] The method of claim 1, further comprising: providing a run-time component configured with transformation instructions to transform each abstract operation, comprising logical fields selected from the plurality of logical fields, into a physical operation consistent with the physical data; and providing a fee calculator configured to calculate a fee for executing physical operations based on the fee schedules.

[004] The method of claim 1, wherein the abstract operation is an abstract query comprising at least two logical fields, the method further comprising: accessing a corresponding fee schedule for each of the at least two logical fields; determining a per request fee for a first one of the at least two logical fields, wherein the per request fee is charged for each abstract operation involving the first one of the at least two logical fields; and determining a per item fee for a second one of the at least two logical fields, wherein the per item fee is charged for each instance of the second one of the at least two logical fields involved in a given abstract operation.

[005] The method of claim 4, further comprising multiplying the per item fee by a number of instances of the second one of the at least two logical fields to determine a product, and summing the product and the per request fee to determine the fee to be charged.

[006] The method of claim 1, wherein at least one fee schedule defined by the abstract model specifies a first fee for a first type of operation and a second fee for a second type of operation; and further comprising calculating the fee to be charged based on the type of operation performed.

[007] A method of providing fee-based access to physical data comprising a plurality

of physical entities each comprising a plurality of physical fields, the method comprising: providing an abstract model for defining abstract operation specifications logically describing operations to access the data, the abstract model comprising: (a) a plurality of logical fields; (b) a mapping rule for each of the plurality of logical fields, which map each of the plurality of logical fields to at least one of the physical entities of the data; (c) a plurality of model entity definitions, each comprising at least one logical field corresponding to a physical field of a physical entity; and (d) a logical field fee schedule for each of the plurality of logical fields, wherein the fee schedules each specify a fee for accessing a corresponding physical field as part of a physical operation.

[008] The method of claim 7, further comprising: transforming, according to the abstract model, abstract operation specifications into physical operation specifications consistent with the physical data, wherein each abstract operation specification includes at least one of the plurality of model entity definitions; and calculating a fee for executing physical operations based on the fee schedules.

[009] A method of providing fee-based access to data comprising a plurality of physical entities, each comprising a plurality of physical fields, comprising: receiving instructions to perform an operation for accessing the data; performing the operation; determining field-specific fees for each of a plurality of the physical fields accessed by the operation; and calculating a total fee to be charged to a user for the operation.

[010] The method of claim 9, wherein the physical entities are database tables.

[011] The method of claim 9, wherein determining field-specific fees comprises determining whether a field-specific fee is a per request fee or a per item fee, wherein the per request fee is a singular fee charged for the operation regardless of a number of instances a corresponding physical field is included in the operation and wherein the per item fee is charged for each instance of a corresponding physical field included in the operation.

[012] The method of claim 9, wherein determining field-specific fees comprises accessing fee schedules for each respective physical field accessed by the operation.

[013] The method of claim 9, wherein determining field-specific fees comprises accessing an abstract model for logically defining the operation accessing the data, the abstract model comprising: (i) a plurality of logical fields; (ii) a mapping rule for each of the plurality of logical fields, which map the plurality of logical fields to physical entities of the data; and (iii) a fee schedule for each of the plurality of logical fields, wherein each fee schedule for a given logical field defines a fee to be charged when the given logical field is involved in an abstract

operation to access a physical entity corresponding to the given logical field.

[014] The method of claim 13, wherein each fee schedule defines at least one of a per request fee and a per item fee, wherein the per request fee is a singular fee charged for the operation regardless of a number of instances a corresponding logical field is included in the operation and wherein the per item fee is charged for each instance of a corresponding logical field included in the operation.

[015] The method of claim 13, further comprising: transforming, according to the abstract model, abstract operations into physical operation consistent with the physical data, wherein each abstract operation includes at least one of the plurality of model entity definitions; and calculating the fee for executing physical operations based on the fee schedules.

[016] A computer-readable medium containing a program which, when executed by a processor, performs operations for accessing physical data comprising a plurality of physical entities, each having a plurality of physical fields, the operation comprising: receiving instructions to perform an operation accessing the data; causing performance of the operation; determining field-specific fees for each of a plurality of the physical fields accessed by the operation; and calculating a total fee to be charged to a user for the operation.

[017] A method for constructing abstract queries defined by a plurality of logical fields which map to a plurality of physical entities of physical data having a particular physical data representation in a database, the method comprising: receiving user input via a user interface, the input comprising a reference to a model entity definition comprising: (i) two or more logical fields each corresponding to a separate physical entity; and (ii) a fee schedule for accessing physical entities based on the model entity definition; based on the model entity definition, selectively adding at least one of the two or more logical fields to an abstract query; receiving a plurality of abstract query contributions for the abstract query, wherein the plurality of abstract query contributions are defined by selected logical fields and a corresponding value for each of the selected logical fields; receiving a plurality of result fields for the abstract query, wherein the plurality of result fields is defined by selected logical fields; converting the abstract query into a physical query consistent with the particular physical data representation of the data; executing the physical query; and calculating, on the basis of the fee schedule, a fee to charge for execution of the physical query.

[018] The method of claim 17, wherein selectively adding the at least one of the two or more logical fields comprises: determining whether the at least one logical field is a required field as specified by the model entity definition; and if so, adding the at least one logical field to the abstract query.

[019] The method of claim 17, wherein a per request fee is defined for a first one of the two or more logical fields and a per result fee is defined for a second one of the two or more logical fields and wherein calculating the fee comprises: (i) calculating a product of the per result fee and a number of results for the second one of the two or more logical fields; and (ii) summing the product and the per request fee.

[020] The method of claim 17, wherein converting the abstract query into the physical query comprises mapping each of the logical fields of the abstract query to respective physical entities of the physical data.

[021] A method for modifying physical data comprising a plurality of physical entities and having a particular physical data representation in a database, the method comprising: receiving a selection of an abstract modification operation; receiving a selection of a model entity definition on which to perform the abstract modification operation, the model entity definition comprising two or more logical fields each corresponding to a separate physical entity; based on at least the received selections, generating at least two physical modification statements, each modifying one of the two separate physical entities of the physical data; ordering the at least two physical modification statements; executing modification operations according to the physical modification statements, whereby the data is modified; and calculating a fee to charge for executing the modification operations based on a defined fee schedule for the model entity definition.

[022] The method of claim 21, wherein the ordering is done according to a physical entity relationships specification defining hierarchical relationships between physical entities of the data to ensure integrity of the data.

[023] The method of claim 21, wherein building the at least two physical modification operations is done according to mapping rules which map the two or more logical fields to their corresponding physical entity.

[024] A method of providing a logical framework for defining abstract operations for accessing physical data comprising a plurality of physical entities each comprising a plurality of physical fields, the method comprising: providing an abstract model for defining abstract operation specifications logically describing operations to access the data, the abstract model comprising: (a) a plurality of logical fields; (b) a mapping rule for each of the plurality of logical fields, which map each of the plurality of logical fields to at least one of the physical entities of the data; (c) a plurality of model entity definitions, each comprising at least one logical field corresponding to a physical field of a physical entity; and (d) model entity fee schedules for each of the plurality of model entity definitions,

wherein the fee schedules each specify a fee for accessing a physical field of the corresponding model entity definition; and providing a run-time component to transform, according to the abstract model, abstract operation specifications into physical operation specifications consistent with the physical data, wherein each abstract operation specification includes at least one user-selected model entity definitions of the plurality of model entity definitions.

[025] The method of claim 24, wherein the abstract operation specification is an abstract query, and further comprising: receiving, via a user interface, the abstract query comprising a plurality of query conditions, result fields and a selection of one of the model entity definitions; accessing the model entity definition corresponding to the selection; determining whether the model entity definition corresponding to the selection specifies one or more required result fields; and if so, adding the one or more required result fields to the query.

[026] The method of claim 24, further comprising transforming, by the run-time component transforms and according to the abstract model, a single abstract operation specification into at least two separate physical operation specifications consistent with the physical data, wherein each physical operation specification modifies a different physical entity of the data and wherein each physical operation specifications is ordered for execution according to a physical entity relationships specification defining hierarchical relationships between the physical entities of the data.

[027] The method of claim 24, further comprising: issuing, by a requesting entity, a request to execute a single abstract operation specification; and transforming, by the run-time component, the single abstract operation specification into the at least two physical operation specifications for modifying the data.

[028] The method of claim 27, wherein transforming the single abstract operation specification into the at least two physical operation specifications comprises: generating the at least two physical operation specifications; and ordering the at least two physical operation specifications according to a physical entity relationships specification of the abstract model.

[029] The method of claim 24, wherein each mapping rule comprises an access method for one of the plurality of logical fields, wherein the access method describes a location of the physical entities of the data.

[030] The method of claim 24, further comprising a logical field fee schedule for each of the plurality of logical fields, wherein the fee schedules each specify a fee for accessing a corresponding physical field as part of a physical operation specification.

[031] A computer-readable medium containing a program which, when executed by a

processor, provides a logical framework for defining abstract query operations, the program comprising: an abstract model for defining abstract queries logically describing operations to query the data, the abstract model comprising: (i) a plurality of logical fields; (ii) a mapping rule for each of the plurality of logical fields, which map the plurality of logical fields to physical entities of the data; and (iii) a fee schedule for each of the plurality of logical fields; a run-time component configured with transformation instructions to transform an abstract query, comprising logical fields selected from the plurality of logical fields, into a physical query consistent with the physical data; and a fee calculator configured to calculate a fee for executing physical queries based on the fee schedules.

[032] A computer comprising a memory and at least one processor, and further comprising a logical framework for defining abstract modification operations for modifying physical data, the logical framework comprising: an abstract model for defining an abstract modification specification logically describing an operation to modify the data, the abstract model comprising: (i) a plurality of logical fields; (ii) a mapping rule for each of the plurality of logical fields, which map the plurality of logical fields to physical entities of the data; and (iii) a fee schedule for each of the plurality of logical fields; a run-time component to transform an abstract query, comprising logical fields selected from the plurality of logical fields, into a physical query consistent with the physical data; and a fee calculator configured to calculate a fee for executing physical queries based on the fee schedules.

[033] The computer of claim 31, wherein each fee schedule defines at least one of a per request fee and a per item fee, wherein the per request fee is a singular fee charged for the operation regardless of a number of instances a corresponding logical field is included in the operation and wherein the per item fee is charged for each instance of a corresponding logical field included in the operation.

[034] A method of providing fee-based access to data comprising a plurality of physical entities, each comprising a plurality of physical fields, comprising: receiving, via a user interface, user input comprising instructions for an operation for accessing the data selected fields of the plurality of the physical fields; determining field-specific fees for each of the selected fields; calculating a fee to be charged to a user for accessing the selected fields; and displaying the fee to the user via a user interface.

[035] The method of claim 34, wherein determining field-specific fees comprises accessing fee schedules for each respective physical field accessed by the operation.

[036] A method for displaying fee information for fee-based access to data comprising a plurality of physical entities, each comprising a plurality of physical fields, comprising: displaying one or more user interface screens for construction of queries; receiving, via the one or more user interface screens, user input defining a query configured to access selected fields of the plurality of physical fields; and displaying, via the one or more user interface screens, a field-specific access fee for each of the selected fields.