SUPPORT FOR MULTIPLE CONTENT-MANAGEMENT DATA MODELS

Field of the Invention

The present invention relates to data structure models, including multimedia data structure models, and facilitates access, retrieval, and data conversion for content management applications. The data structure model is used for organizing, inter-relating, and accessing data and files, including relational, network, hierarchical, and entity-relationship models, and graphics, text, and web page files, among others.

Background of the Invention

Multimedia data management and delivery is a complex, computer and memory intensive, task. As used herein, "multimedia" data refers to combinations of sound, graphics, animation, text, and video data, which may be linked in an associative system of storage and retrieval, which may be interactive, and which may be linked to other media.

A multimedia content management system needs to manage a large volume of disparate data, including unstructured data as well as structured data. The structured data often include "metadata" that describe the unstructured data so that the latter can be organized, grouped and correlated in various ways, searched, retrieved, and administered. Within a content management system, a relational database management system (RDBMS) is often used to manage these structured data so that off-the-shelf RDBMS data management technologies can be employed. However, the type of the unstructured data, that is, multimedia content, differs from application to application.

Because the type of data (including its schema, metadata, applicable extenders, and the like) differs between applications and content data models (as streaming, graphical, visual, video, audio, text, numeric), a different database design is needed to support the metadata and schema database of each application and data model. Heretofore, it was not possible to support different content management applications using different content data models with a single, general-purpose content management system. Such a system could not have a pre-designed, i.e., "one size fits all," database management system. Rather, the content management system must have been provided with the capability of allowing an application developer to describe the required structured data so that

a database with a suitable design could be created for that application. This description was normally expressed in the abstraction of a high-level data model supported by the content management system, usually through an application programming interface (API) or a graphical user interface (GUI). While this high-level data model provided the flexibility to support more than one application, it also limited the coverage to only those applications that could be expressed in that particular high level model.

The data model is an essential aspect of a content management system because it determines the potential capability of the system as well as its limitations. However, while a data model can sometimes be expanded somewhat, such an expansion is usually very limited and will not be able to capture a different paradigm, especially one with an incompatible or even conflicting concept.

Thus, a clear need exists for an extensible and scalable data model that readily supports a wide range of disparate content data models, and which can be readily implemented in a system or program product.

A further need exists for a tool, tool kit, tool set, or wizard to create and populate a relational database of disparate content schema and metadata, for an extensible and scalable content management RDBMS.

Summary Of The Invention

Accordingly, the present invention provides a method of representing a multimedia content management object in a database. In accordance with the method of the present invention a database is employed, comprising one or more high level content models and a low level physical model of the multimedia content data. The low level physical model provides a mapping to a data engine. The method includes the steps of entering multimedia data metadata and schema in the low level physical representation, and mapping the metadata and schema to the data engine. The data engine is preferably a relational database management system, an object oriented database management system or an XML data repository.

The method preferably provides for the mapping to the data engine further comprising a method of representing the multimedia content management object as an item in a relational database adapted for representing multimedia content management data in one or more relational database tables. Each of the relational database tables have at least one

row with a plurality of columns. The method comprises the steps of associating a root component of the object as an item to a row in a first relational database table; associating attributes of the root component to corresponding columns of the first relational database table; associating additional components of the object, if any, to rows in the additional relational database tables; and using the item as a building block to construct a plurality of high level content management data models.

Accordingly, the present invention further provides a computer program product comprising instructions which, when executed on a data processing system having a non-volatile memory storage device, causes the system to carry out the method described above.

Accordingly, the present invention further provides a system for managing and delivering multimedia data object items from a multimedia data object content repository through a multimedia data object content server to a client. The multimedia data object comprises multimedia data object items, each comprising multimedia data object attributes and components. The multimedia data object content server is controlled and configured to associate a root component of the object to a row in a first relational database table; associate attributes of the root component to corresponding columns of the first relational database table; and associate additional components of the object, if any, to corresponding rows in additional relational database tables. The item is used as a building block to construct a plurality of high level data models.

The Figures

A preferred embodiment of the invention will now be described in detail by way of example only with reference to the following drawings:

FIGURE 1 illustrates a multimedia database system with multiple clients, a content management server, and multiple content servers.

FIGURE 2 illustrates the layered structure of the method, system, and program product of the invention, with a structure of the API's as high level data models, a generic, low level infrastructure as the physical data model, and the underlying Relational Database Management System.

FIGURE 3 illustrates a simplified overview of a Relational Database Management System of Figure 2 showing hierarchal RDBMS tables including an Item Table with a row for the root component and including component and attribute cells (columns), component tables with sub-component and attribute cells (columns), and grandchild tables for both sub-components and attributes, with suitable pointers.

FIGURE 4 depicts the relationships between the RDBMS tables of Figure 3, and illustrates a side-by-side mapping of an Item type, including the root table, child tables, and grand child tables, mapped to an Item, as an instance of the Item type, including the root components, child components, and grand child components. Each component is a row in the corresponding tables.

FIGURE 5 illustrates a row in a table of the RDBMS of Figure 2, relating source items, target items, and link items.

FIGURE 6 illustrates the application of resource references to target items.

FIGURE 7 illustrates a personal identification application having multimedia content stored at various servers for use at a terminal, for example in a security system.

Detailed Description of the Invention

As shown in FIGURE 1, a multimedia system is a complex client-server system with multiple clients, 11a, 11b, a server, 13, and one or more content servers, 17a, 17b, each with one or more content stores, 19a, 19b, 19c, 19d. It should be noted that the content stores may contain structured data, such as metadata for content, or unstructured data, such as textual data, image data, streaming audio, or streaming video. These data may be, and typically are, in a variety of disparate formats.

The server, 13, which may include a web server, also includes the content manager server. The content manager server is the repository of the content management RDBMS described herein. While the present invention is illustrated with respect to relational database management systems, alternative data repositories such as object oriented database management systems or XML data repositories may be used.

To accommodate more than one content data model in a high-level data model, an extensible content management system which may be used to introduce appropriate metadata and schema and new APIs to support new high-level data models. This extensibility is enabled by an architecture that is built around a low-level infrastructure supporting a physical data model and implemented through a hierarchical structure of RDBMS tables of content and content metadata, as items, components, sub-components, and attributes, with corresponding rows and columns (attributes) within the rows. The architecture and an associated infrastructure and physical data model are used as building blocks to support high-level data models. This physical data model is independent of application-specific semantics, maps directly and efficiently to a relational database designed to exploit RDBMS capabilities, and can be enriched over time to expand its functions and/or to exploit new RDBMS capabilities.

The design point for this physical data model is different from that of a high-level data model exposed to an application developer. The high-level data model focuses heavily on its logical aspects (its usability and expressiveness with respect to a class of applications), whereas the physical data model focuses heavily on its physical aspects (storage access efficiency and ability to exploit RDBMS capabilities) to provide generic, basic functions broadly useful to many content management applications. This physical data model, or any part of it, may or may not be exposed to user through a high-level data model.

Conceptually, as shown in FIGURE 2, the method, system, and program product of the present invention utilize a high level content model that is able to support a diverse and open set of content application requirements and a plurality of high level data models, and a low level physical model of the data content that is extensible to accommodate new technologies, new standards, and new requirements and provides efficient mapping to the data engine (a database management system, as a relational database management system or an object oriented database management system, by way of example and not limitation). To be noted is that each high level data model is in the form of an Application Program Interface (API) embodying a representation of data structure and constraints. Also to be noted is that the low level data model or infrastructure, 25, can support multiple high level models, 21, 22, 23. The extensible and scalable content management system of the invention contains one or more APIs supporting disparate high level data models, 21, 22, 23, supported by a generic, low level infrastructure, 25, that is, a physical data model,

which in turn is supported by the Relational Database Management System (RDBMS), 27.

FIGURES 3, 4, 5, and 6 illustrate the relationships between the physical model and its representation in a RDBMS environment. In a relational database data is represented as tables. The building blocks of a relational database table are tables of rows and columns (attributes). Specifically, a table consists of a row of column headings together with zero or more rows of data values. For a given table (1) the column heading specifies one or more columns, and (2) each data row contains exactly one value (or a null value) for each one of the columns specified in the column heading row.

The primary, standalone unit of content managed by the physical layer of the content management system of the invention is an "Item." FIGURE 3 shows an Item Database, 30, with a root component, 31, having Item Rows, 32a and 32b. An item consists of a typed hierarchical structure of components, 35, and one or more levels of sub-components, 38a, 38b, with attributes 33a, 33b, 33c, 33d, 34a, 34b, 34c, and has a unique identifier "ItemID".

Each component, 35, contains a reference, 36a, to the parent component. The reference is through a foreign key.

The Sub-component tables, 38a and 38b, may reference further sub-components or attributes. The sub component table, 38b, is illustrated as referencing pointers to content data repositories, 39a, and 39b.

FIGURE 4 represents a more general, higher level view of the model shown in FIGURE 3. Through the high-level data model, or within an application, an "Item", that is, a multimedia content management object item, can be used to represent a real object, as a document, a folder, a media asset, or other real-world content entity. FIGURE 4 illustrates a mapping of hierarchal physical items and attributes thereof, 41b, on the right, to hierarchal RDBMS representation, that is, RDBMS Item Types, 41a, on the left. An "ItemType," is a user defined schema and a named hierarchical structure of RDBMS tables, each of which is called a "ComponentType," 35, of the "ItemType" Each "ComponentType" contains user-specified attribute(s) as well as certain system attributes.

A user-specified attribute can be of any data model supported by the RDBMS, or can be of a special data model supported by the content

management system. An "Item" logically consists of a row in the root table of the respective "ItemType," and any number of rows in each child table of the "ItemType," with a condition that the parent of a child row contained in an "Item" must also be contained in the "Item." An "Item" is a hierarchy of rows that are stored in a hierarchy of tables. Each of these rows is called a "Component" of the "Item" and has a "ComponentID" that uniquely identifies the "Component" within the respective "ComponentType."

Cardinality constraints can be defined for each child "ComponentType" to assure a minimum and/or a maximum number of rows of that type under each of their parent rows. The default minimum is zero and the default maximum is infinity. An "ItemType" can also be ordered or unordered. For an "Item" of an ordered "ItemType," the child rows under each parent row are ordered, and the ordering is maintained by the system.

An "ItemType" can be versioned. If so, multiple versions of an "Item" are maintained for every "Item" in the "ItemType." They share the same "ItemID" but each has a different version number. The number of latest versions to be retained for an "Item" can be specified for each versioned "ItemType." The default is infinity, i.e., no automatic purging of old versions will be performed by the system.

In one preferred embodiment, the parent-child relationship between two "ComponentTypes" within an "ItemType" is maintained by a RDBMS Foreign Key created in the child "ComponentType" (or sub-component) referencing the parent "ComponentType." Thus, the referential integrity of each child "Component" (or sub-component) to its parent "Component" is assured by the RDBMS. In this embodiment, each "Component" in an "Item" contains at least the following system attributes: "ItemID", "ComponentID", and ParentID, and optionally may contain an "Item" VersionNumber, and ACL Code (Access Control List code). Although "ItemID" , "Item" VersionNumber, and ACL Code are only needed for the root "Component," they may be included in every "Component" to facilitate access. Other system attributes can also be included to facilitate access or maintenance, e.g., "ComponentType," the number of immediate child Components underneath a "Component," etc. Also a separate RDBMS table is created for each "ComponentType" defined for an "ItemType;" i.e., tables are not shared between two "ItemTypes". This design simplifies the logic and also enhances database scalability. In a different embodiment, a table can be shared among several "ItemTypes" to store Components (rows) of similarly defined ComponentTypes for these several "ItemTypes". In this case, including "ItemType" as a system

attribute in this shared table can facilitate access. Sharing a table in this manner allows a faster search when the search scope covers multiple "ItemTypes" that contain common attribute(s) being searched on.

As illustrated in FIGURE 5, Items (including Child Components, Grand Child Components and Attributes) can relate to or be associated with each other, for example through a system table, using Links. A Link is an "external" relationship that binds a source "Item," 51, to a target "Item," 53, along with a LinkType, 55, and an optional Link "Item," 57. This relationship is "external" since it is not imbedded in either the source or the target "Item". The LinkType is a categorization label so that an application can selectively traverse, or search on, Links of a particular type. A Link "Item" is a third "Item" that can be used to describe this relationship if needed. Links can be used to support one-to-many and many-to-many relationships, including aggregation and containment relationships.

The semantics of each relationship are defined by a high-level data model. One example is "foldering", in which case an "Item" representing a folder is Linked to the Items (representing documents or other folders) that are "contained" in this folder. For example, Source Item 51, acts as a container for Target Item, 53. The binding of a Link to the source or the target "Item" can be either version-specific or version-independent. The former binds a specific version of an Item, whereas the latter binds to any/all versions of an "Item". In one embodiment, Links are maintained as rows in one or more dedicated RDBMS tables containing at least the following attributes: source "ItemID" , source "Item" VersionNumber, target "ItemID" , target "Item" VersionNumber, LinkType, Link "ItemID", and Link "Item" VersionNumber.

As illustrated in FIGURE 6, items can also relate to each other using "Item Reference" attributes, which are maintained like other attributes contained in an "Item". An "Item Reference" attribute is a user-defined attribute created as a part of a "ComponentType," 36, within an "ItemType," 32. This attribute has a special data model maintained by the content management system. It contains a reference, 58, that points to a target "Item," 32'. Similar to a Link, (FIGURE 5) the binding of the reference, 58, to the target "Item," 32', can be either version-specific or version independent. When an "Item Reference," 58, attribute is specified, a Scope of Reference and a Delete Semantics are defined for the referenced Items. The scope specifies which "ItemType" (s) the referenced Items can belong to, whereas the delete semantics are similar to the

delete rules maintained by a RDBMS to assure referential integrity when a referenced "Item" is to be deleted. An "Item Reference" attribute, 58, can also be unscoped, in which case a referenced "Item" can belong to any "ItemType." In a preferred embodiment, a homogeneously scoped "Item Reference" attribute (i.e., when the scope is a single "ItemType") is supported by a RDBMS Foreign Key. Heterogeneously scoped (i.e., when the scope includes multiple "Item" Types) and unscoped "Item Reference" attributes are maintained by the content management system using RDBMS triggers.

Another special data model supported by this physical data model is that of a Resource Reference attribute, 59, also illustrated in FIGURE 6. As illustrated in FIGURE 6, a Resource Reference, 59, contains a reference to a Resource, 61, managed by a Resource Manager of the content management system. A Resource Manager is a special-purpose server that manages resources of a particular type. Examples are an Object Server, which manages a large volume of binary objects, and a Stream Server, which manages and delivers streaming objects (e.g., audio and video content). The extensible content management system is able to accommodate new Resource Managers, each of which is specially designed to manage a specific type of resources, with its specific representation and methods that are not necessarily modeled by the physical data model described here. The unstructured data managed by the content management system are largely managed internally by one or more of these Resource Managers. The integrity of a Resource Reference is maintained by the content management system.

The physical data model described herein is able to support a wide range of high-level data models to cover a diverse spectrum of application domains, as well as to support many industry standards (e.g., Dublin Core, XML). Equally important, this physical data model also provides enough flexibility to allow database optimization so as to support high-level data models efficiently. In one example, a high-level data model uses Items of one "ItemType" to represent standalone, sharable images, each of which has its own attributes, versions, and ACL. The actual images are stored as Resources. Items of another "ItemType" are then used to represent documents, and Links are used to bind each document to the images it contains. In another example, to support a high-performance production application, a single "Item" is used to represent a document together with the images contained in the document. (The actual images are also stored as Resources.) In this case, performance is critical but there is no sharing requirement for the images.

"ItemView(s)" can be defined on an "ItemType." An "ItemView" is a named template to restrict access to an "ItemType" to certain "Component" tables and attributes. It is essentially a set of RDBMS views for the corresponding "Component" tables in the "ItemType."

One application of the present invention is a multimedia distributed database management system with distributed card reader access, for example, as would be used in a magnetic card security system, a credit card or debit card or automatic teller machine card system, or an airline frequent traveler system. One version of this system is shown in FIGURE 7. The system, 70, is actuated by identification indicia, e.g., encrypted data, on a magnetic or optical card, 70b, read by a card reader 70a.

After reading the card, 70b, biometric readings are taken. Exemplary biometric data includes fingerprints at finger print reader 70c, voice prints and samples, facial images, and eye images, at digital camera, 70d, among others. The biographical, audio, and image data are typically accessed through and processed by separate servers 72a, 73a, 74a, and 75a, and stored in separate databases, 72b, 73b, 74b, and 75b, for example, fingerprint data, voice data, and facial and eye image data are stored as multimedia data objects on separate fingerprint data, voice data, and facial and eye image databases, with such attributes as data models (images, mathematical representations of image data, audio data, and mathematical representations of audio data), as well as repository addresses (e.g., URL's or IP addresses), where the data is the item and the addresses and definitions of data models are the attributes, and the image presented to the screener, security guard, cashier, boarding agent, or enforcement professional is the resource.

The method and system are physically implemented in a program product. The program product may reside on one computer or on several computers or on a distribution server or a disk or disks or tapes. The program itself may be encrypted and/or compressed, as would be the case of distribution media or a distribution server, for example, before installation or it may be installed on the one or more computers.

A further application of the invention is a method and program product in the nature of a wizard (that is, a structured series of dialogs that ask questions and use the answers or choices to produce a result) for populating the multimedia content management system with content schema and metadata, including high level content models and low level physical

models. This is accomplished by presenting a query to a user as to a content item, and, based upon the end user's response, presenting one or more subsequent queries as to the content item, and based upon the responses, determining the sub-components and attributes of the item.

CLAIMS

1. A method of representing a multimedia content management object in a database comprising one or more high level content models and a low level physical model of the multimedia content data, said low level physical model providing a mapping to a data engine, said method comprising:

a. entering multimedia content data metadata and schema in the low level physical representation, and
b. mapping the metadata and schema to the data engine.

2. The method of claim 1 wherein the one or more high level content models comprises an application program interface embodying a representation of data structure and constraints.

3. The method of claim 1 wherein the data engine is chosen from the group consisting of relational database management systems, object oriented database management systems, object-relational database management systems and XML data repositories.

4. A method of claim 1 wherein the mapping to a data engine comprises a method of representing the multimedia content management object as an item in a relational database adapted for representing multimedia content management data in one or more relational database tables, each of said relational database tables having at least one row with a plurality of columns, said method comprising:

a. associating a root component of the object as an item to a row in a first relational database table;
b. associating attributes of the root component to corresponding columns of the first relational database table;
c. associating additional components of the object, if any, to rows in additional relational database tables; and
d. wherein the item is used as a building block to construct a plurality of high level content management data models.

5. The method of claim 4 further comprising a method of managing a multimedia content management system comprising the multimedia content management object and the relational database.

6. The method of claim 4 or claim 5 wherein each of the additional components comprises a child component of a root component or a child component of another component.

7. The method of claim 4 or claim 5 wherein an attribute comprises a pointer to a data resource stored in a separate repository.

8. The method of claim 4 or claim 5 wherein a row in a table comprises a link between a source item and a target item.

9. A computer program product comprising instructions which, when executed on a data processing system having a non-volatile memory storage device, causes said system to carry out a method as claimed in any one of claim 1 to claim 8.

10. A system for managing and delivering multimedia data object items from a multimedia data object content repository through a multimedia data object content server to a client, wherein the multimedia data object comprises multimedia data object items, each of the multimedia data object items comprising multimedia data object attributes and components, and wherein the multimedia data object content server is controlled and configured to:

a. associate a root component of the object to a row in a first relational database table;
b. associate attributes of the root component to corresponding columns of the first relational database table; and
c. associate additional components of the object, if any, to corresponding rows in additional relational database tables;

wherein the item is used as a building block to construct a plurality of high level data models.