(12) STANDARD PATENT (11) Application No. **AU 2006333375 B2**
(19) AUSTRALIAN PATENT OFFICE

(54) Title
Method and mechanism for loading XML documents into memory

(51) International Patent Classification(s)
G06F 17/22 (2006.01)

(21) Application No: **2006333375** (22) Date of Filing: **2006.11.29**

(87) WIPO No: **WO07/078479**

(30) Priority Data

(31) Number	(32) Date	(33) Country
11/317,101	**2005.12.22**	**US**

(43) Publication Date: **2007.07.12**
(44) Accepted Journal Date: **2011.06.23**

(71) Applicant(s)
Oracle International Corporation

(72) Inventor(s)
Jain, Namit;Murthy, Ravi;Chandrasekar, Sivasankaran;Agarwal, Nipun

(74) Agent / Attorney
Pizzeys Patent and Trade Mark Attorneys, Level 20 324 Queen Street, Brisbane, QLD, 4000

(56) Related Art
US 2005/0102256 A1

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
12 July 2007 (12.07.2007)

PCT

(10) International Publication Number
WO 2007/078479 A3

(51) International Patent Classification:
G06F 17/22 (2006.01)

(21) International Application Number:
PCT/US2006/045700

(22) International Filing Date:
29 November 2006 (29.11.2006)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
11/317,101 22 December 2005 (22.12.2005) US

(71) Applicant *(for all designated States except US)*: **ORACLE INTERNATIONAL CORPORATION** [US/US]; 500 Oracle Parkway, Redwood City, CA 94065 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **CHANDRASEKAR, Sivasankaran** [IN/US]; 540 Everett Avenue, Palo Alto, California 94301 (US). **AGARWAL, Nipun** [US/US]; 4766 Cheeney Street, Santa Clara, California 95054 (US). **JAIN, Namit** [IN/US]; 2234 Glanera Street, Santa Clara, California 95054 (US). **MURTHY, Ravi** [IN/US]; 33227 Jamie Circle, Fremont, California 94555 (US).

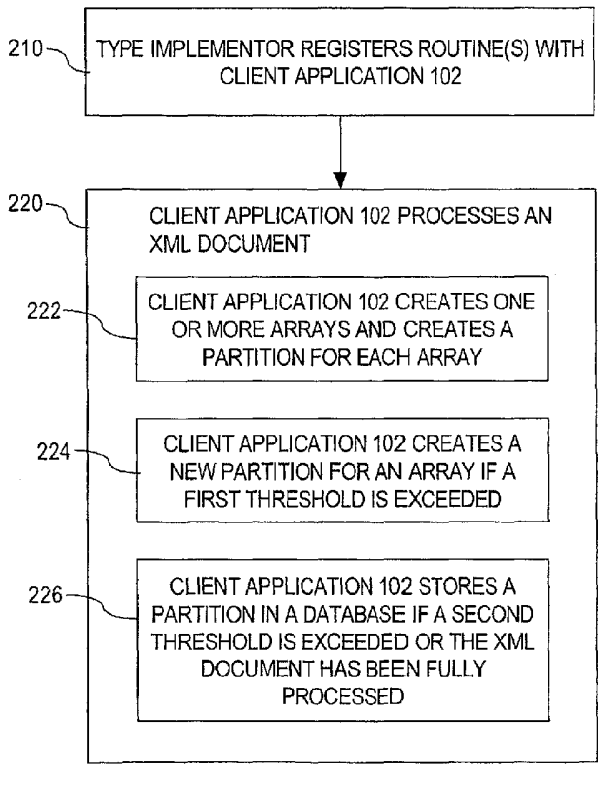
(74) Agents: **BROKAW, Christopher_** et al.; HICKMAN PALERMO TRUONG & BECKER LLP, 2055 Gateway Place, Suite 550, San Jose, CA 95110 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LV, LY, MA, MD, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, SV, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM),

[Continued on next page]

(54) Title: METHOD AND MECHANISM FOR LOADING XML DOCUMENTS INTO MEMORY



(57) Abstract: A method and apparatus for loading an XML document into memory is provided. A client loads one or more array elements into a first partition of an array that is maintained in memory. Each array element represents an XML element of an XML document. Upon determining that an amount of data maintained in the first partition exceeds a first threshold, the client subsequently loads array elements into a new partition of the array. Upon determining that an amount of data maintained in the memory of the client exceeds a second threshold, the array elements of the least recently used partition are persistently stored in a database without persistently storing the entire XML document. When the last XML element of the XML document is loaded into a partition of the array, that partition is persistently stored in the database, thereby causing the entire XML document to be stored in the database.

WO 2007/078479 A3

European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*
— *with amended claims*

(88) Date of publication of the international search report: 13 September 2007

Date of publication of the amended claims: 25 October 2007

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

METHOD AND MECHANISM FOR LOADING XML DOCUMENTS INTO MEMORY

FIELD OF THE INVENTION

[0001] The present invention relates to database systems, and in particular, to techniques for loading XML documents into memory.

BACKGROUND

[0002] The approaches described in this section are approaches that could be pursued, but not necessarily approaches that have been previously conceived or pursued. Therefore, unless otherwise indicated, it should not be assumed that any of the approaches described in this section qualify as prior art merely by virtue of their inclusion in this section.

[0003] Structured data conforms to a type definition. For example, a type definition for a "person" type may define distinct attributes such as "name," "birthdate," "height," "weight," and "gender." Each "instance" of a particular type comprises a separate value for each of the attributes defined by the particular type. For example, an instance of the "person" type might comprise values such as "Fred Brown," "January 1, 1980," "72 inches," "240 pounds," and "male." Each attribute is also of a type. For example, the "name" attribute might be of a "string" type, the "birthdate" attribute might be of "date" type, and the "gender" attribute might be of an "enumerated" type. Structured data might comprise multiple different instances of the same type.

[0004] Different approaches may be used to store structured data into a database. One such approach is called "statement-based path loading." According to the statement-based path loading approach, a client application parses structured data that comprises one or more instances of a type. Values within the structured data correspond to attributes of the type. The client application generates Structured Query Language (SQL) statements, such as INSERT commands, that, when executed by a database server, cause the database server to insert the values into corresponding columns of a database table. Unfortunately, due to its heavy use of the SQL engine, statement-based path loading often suffers in terms of performance and memory consumption.

[0005] Another approach for storing structured data into a database is called "direct path loading." Through direct path loading, values within structured data are stored directly into a database without causing the SQL engine to load each row of data. By consulting a control file that is associated with the structured data, a client application can determine the data type of instances of structured data. If the structures of the types are defined to the client application, then, based on those structures, the client application can create an array that

corresponds to the types' attributes. The client application can populate the array with values that correspond to that attribute. Once the array is populated, the client application can convert the array into a stream of data that conforms to the format of a database's data blocks. The client application then can stream the data to a database server, which can write the data directly into one or more data blocks in the database. Direct path loading exhibits performance superior to that of statement-based path loading.

[0006] Some types indicated by a control file may be standard types that are defined to a client application, e.g., a scalar type is an example of a standard type. However, some types indicated by a control file might not be among the types that are defined to the client application. Types that are not defined to a client application are called "opaque types" relative to the client application, because the internal structure of such types is obscured from, or unknown to, the client application. The internal structure of an opaque type, including the number and types of attributes of the opaque type, often are defined only to a program that implements the opaque type. An opaque type implementor may be external to both the client application and the database server.

[0007] An opaque type may be an XML type. An example of an XML type is provided in co-pending U.S. Patent Application No. 10/259,278. An XML schema is metadata that describes a hierarchical structure. Instances of the XML schema comprise data that conforms to the structure described by the XML schema. Through XML elements expressed in the structure, an XML schema defines one or more types.

[0008] An XML document is a document that contains one or more XML elements that conform to an XML schema. Unfortunately, the amount of memory required to maintain an array representing the XML elements of an XML document may be large. Further, maintaining the control file of an XML document in memory also requires a significant amount of memory, e.g., in some cases the amount of memory required to maintain a control file for an XML document may be ten times the amount of memory to maintain the corresponding XML document in memory. As a result, a large amount of memory is required by a client application to load an XML document into memory when transferring the XML document to a persistent storage. Moreover, transferring XML documents to persistent storage in this manner is very CPU intensive for the client application, which may result in performance degradation.

[0009] Consequently, an approach for loading XML documents into memory for use in transferring the XML documents to a persistent storage that avoids the aforementioned problems is advantageous.

BRIEF DESCRIPTION OF THE DRAWINGS

[0010] Embodiments of the present invention are illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings and in which like reference numerals refer to similar elements and in which:

[0011] FIG. 1 is a block diagram that illustrates a system in which XML documents may be loaded into memory and transferred to persistent storage according to an embodiment of the present invention;

[0012] FIG. 2 is a flowchart illustrating the functional steps of creating and using partitions according to an embodiment of the invention; and

[0013] FIG. 3 is a block diagram that illustrates a computer system upon which an embodiment of the invention may be implemented.

DETAILED DESCRIPTION

[0014] In the following description, for the purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the embodiments of the invention discussed herein. It will be apparent, however, that the embodiments of the invention discussed herein may be practiced without these specific details. In other instances, well-known structures and devices are shown in block diagram form in order to avoid unnecessarily obscuring the embodiments of the invention discussed herein.

FUNCTIONAL OVERVIEW

[0015] An approach for loading an XML document into memory is provided. According to the embodiments discussed herein, XML elements of a large XML document may be separately loaded into one or more separate logical units (denoted individually herein as a partition) in the memory of a client application, and subsequently the partitions in memory may be processed (for example, a partition may be persistently stored in a database) without processing the entire XML document. In this manner, the amount of memory required by the client application to process the XML document may be minimized, as only a portion of the XML document is maintained in memory at any one time.

[0016] In an embodiment, an entity that implements an XML type registers, with a client application, one or more routines that are associated with an XML type. Using those routines, a client application loads one or more array elements that each represents an XML element of an XML document into a first partition of an array that is maintained in memory. A partition is a logical unit for transferring data from memory to a persistent store, such as a database. According to an embodiment, a partition can hold a portion of a data structure, such as an array, and may maintain information that identifies what portion of the data structure is represented by the partition. For example, a particular partition may contain a

starting point identifier that identifies a first location of the array where the particular partition starts, and an ending point identifier that identifies a second location of the array where the particular partition ends.

[0017] Upon determining that an amount of data maintained in the partition exceeds a first threshold, the client application calls a routine associated with the XML type to create a new partition in the array. Thereafter, the client application loads XML elements of the XML document into array elements of the new partition of the array.

[0018] Upon determining that an amount of data maintained in the memory of the client application exceeds a second threshold, the client application persistently stores the array elements of a partition (such as the least recently used partition) maintained in memory to a database. Unless the entire XML document has been loaded into that single partition, a portion of the XML document is stored without persistently storing the entire XML document. The client application will continue to process the XML elements of the XML document in this fashion until the last XML element is processed. When the last XML element of the XML document is loaded into a partition of the array, the client application persistently stores any partitions maintained in memory to the database, thereby causing the entire XML document to be persistently stored in the database.

[0019] Having described a high-level overview of an embodiment, an overview of the architecture of an embodiment will be described below.

ARCHITECTURAL OVERVIEW

[0020] FIG. 1 is a block diagram that illustrates a system 100 in which XML documents may be loaded into memory and transferred to persistent storage according to an embodiment of the invention. System 100 comprises a client application 102, a database server 104, a database 106, and type implementors 108A-N. Client application 102, database server 104, and type implementors 108A-N are coupled communicatively to each other. Database server 104 is coupled communicatively to database 106.

[0021] Client application 102 reads or otherwise receives semistructured data 118 as input. Semistructured data 118 is data that conforms to alternative defined structures, rather than a single defined structure. Semistructured data 118 comprises instances of a type. For example, semistructured data 118 may comprise one or more XML instances that conform to an XML schema, e.g., an XML document. Semistructured data 118 also comprises an identity of the type. For example, the type may be identified as an XML type. Semistructured data 118 does not indicate the structure of the type. The structure of the type is not defined to client application 102.

[0022] Client application 102 creates an array 116A in client application address space 114. Client application address space 114 comprises a segment of memory allocated for use by client application 102. Client application 102 populates array 116A by adding array elements to the array. Each array element may represent an XML element of an XML document. Array 116A may be populated one partition at a time, as explained in further detail below. Some of the values used to populate array 116A may be specified in semistructured data 118, and other values may be derived from values specified in the semistructured data.

[0023] As explained in the loading opaque types patent, type implementers 108A-N may register routines with client application 102 which, when invoke, perform operations involving array 116A. For example, one or more of the routines may store values in a particular partition of array 116A or may create a new partition. An invoked routine may return, to client application 102, one or more pointers to one or more addresses within client application address space 114 at which one or more populated elements of a partition of array 116A can be found. Using the one or more pointers, client application 102 can locate and read the populated rows of a partition of array 116A. In this way, client application 102 can determine the amount of data loaded into a particular partition.

[0024] One or more of the attributes of the type may be of a nested type that indicates one or more other attributes. For example, a parent type might indicate a first attribute that is of a scalar type, and a second attribute that is of a "purchase order" child type. The "purchase order" type also might indicate several attributes. In this case, the routines associated with the parent type may invoke routines associated with the child type. Client application 102 does not need to be aware of or invoke routines associated with the child type. When a routine is invoked to describe the structure of the parent type, that routine invokes another routine to describe the structure of the child type. When a routine is invoked to create an array for the parent type, that routine invokes another routine to create an array for the child type. When a routine is invoked to populate the array for the parent type, that routine invokes another routine to populate the array for the child type. Each array created maybe associated with one or more partitions, as explained below. To preserve the relationship between the array of the parent type and the arrays of any child types, a set identifier that links a row of the parent array with the corresponding rows of the child array is generated and stored in the parent array. Set identifiers for array elements of a partition may be stored in that partition.

[0025] Array may be populated independently of each other, although only a single partition of any one array will be populated at any one time, as explained below. Arrays may be loaded and streamed asynchronously and independently.

[0026] Based on the populated rows of the partitions, client application 102 generates a data stream. A data stream for a partition of an array may be generated independently of a data stream for another partition of another array. The data stream conforms to the format of data blocks within database 106. As a result, the data stream generated by client application 102 may be written directly to database 106 without causing the SQL engine to load each row of data. Client application 102 streams the data to database server 104. A stream generated based on one array may be sent to the database server independently of a stream generated based on another array. Database server 104 writes the data received from client application 102 directly into one or more data blocks in database 106.

[0027] A data block is an atomic unit of storage that stores the records of database. When a database read or writes records from a database, the smallest unit of data read from or written to persistent store is a database block. Typically, a data block is stored in memory in buffer of buffering system. The data block may not only contain records but also control information used to process data within the data block.

[0028] Additional details regarding how to load XML elements into an array maintained in memory are provided in the loading opaque types patent. Having described an illustrative architecture upon which an embodiment may be implemented, the process of creating and using partitions will be described in further detail below.

CREATING AND USING PARTITIONS

[0029] FIG. 2 is a flowchart illustrating the functional steps of creating and using partitions according to an embodiment of the invention. By performing the functional steps of FIG. 2, the memory requirements to load an XML document into memory may be reduced by establishing certain thresholds that, when exceeded, trigger processing of the partition(s) maintained in memory. Partitions may be persistently stored from memory using the direct path loading technique. As a result, the techniques described herein provide a more scalable approach for loading large XML documents into memory. Further, certain XML documents, that could not previously be loaded into memory due to memory constraints, may be loaded into memory using the techniques of FIG. 2, as only portions of the XML document are loaded into memory at a single time. The functional steps of FIG. 2 shall be explained with reference to FIG. 1.

[0030] In step 210, a type implementor registers one or more routines with client application 102 as described above. For example, type implementor 108A may register routines 110AA-AN with client application 102A. One or more of routines 110AA-AN may perform the following functions: (a) create a new partition within an array maintained in memory by the client application 102, and (b) populate a row representing an XML element

of an XML document into a particular partition of an array. After one or more routines are registered with client application 102, processing proceeds to step 220.

[0031] In step 220, client application 102 processes an XML document. In step 220, client application 102 may processes an XML document by loading XML elements of the XML document into one or more partitions maintained in memory, and persistently storing the partitions from memory to database 106. Partitions may be persistently stored, from memory, prior to the entire XML document being loaded into memory to make additional memory available to load further XML elements of the XML document into a partition in memory. The performance of step 220 will be described in further detail below.

[0032] In step 222, client application 102 may process an XML document by creating one or more arrays in memory by invoking a routine associated with the XML type of the XML document as described above. Multiple arrays may be created for a single XML document. For example, when a routine is invoked to create an array for a parent type having a child type, that routine invokes another routine to create an array for the child type. Once an array is created in memory for the XML document, client application 102 may create a partition for an array by invoking a routine associated with the XML type of the XML document as described above. Client application 102 loads array elements into only one partition of a particular array at a time.

[0033] After the partition(s) and array(s) are created in memory, client application 102 reads an XML element from the XML document. Client application 102 then loads the XML element into an appropriate partition maintained in memory. Client application 102 may then perform steps 224 and 226. After the performance of steps 224 and 226, client application 102 may then repeat the performance (a) loading an XML element from the XML document into a partition in memory, and (b) the performance of steps 224 and 226 until all the XML elements of the XML document have been loaded into a partition in memory. Alternately, client application 102 may perform one or more of steps 224 and 226 after reading two or more XML elements from the XML document into a partition.

[0034] In step 224, client application 102 determines if the amount of data maintained in a particular partition ("the full partition") for an array exceeds a threshold (the "partition size threshold"). If client application 102 determines that the amount of data maintained in the full partition of the array does exceed the partition size threshold, then client application 102 creates a new partition for the array by invoking the one or more routines associated with the XML type of the XML document as described above. Thereafter, when client application loads XML elements of the XML document into the array, client application 102 loads the XML elements into the new partition of the array, rather than the full partition of the array.

[0035] In an embodiment, client application 102 may determine if the full partition threshold has been exceeded for a particular partition using a pointer to the partition returned by the routine, invoked by client application 102, to create the partition. Using the pointer to the partition, client application 102 may locate and read the rows of the partition to determine whether the amount of data maintained in a particular partition exceeds the partition size threshold.

[0036] The partition size threshold may be configured by a user or it may be dynamically determined based on the total amount of memory of client application 102.

[0037] The size of each partition created (either in step 222 or in step 224) may by dynamically determined based on, at least in part, the type of data to be stored in the partition or the location, within database 106, where the data loaded into the partition will be eventually stored. For example, the size of a partition may be based on the presence of repeating elements within the XML document. Repeating elements are multiple instances of the same XML element within an XML document. To illustrate, if an XML document corresponds to a bill, then the line items of the bill may be repeating elements since the line items are multiple instances of the same type of data, in this case items of a bill. It is advantageous to store XML elements corresponding to those line items in a single partition, to minimize the amount of the tables, within database 106, which are required to store data loaded into a partition. Thus, the size of the partition may be determined to ensure that XML data, loaded into the partition, will be stored within the smallest amount of tables in database 106. Also, the size of a partition may be determined based on, at least in part, a number of repeating XML elements within the XML document, thereby increasing the likelihood that all the XML elements loaded into the partition will be stored in the same table or similar manner in database 106.

[0038] Additionally, when client application 102 invokes a routine to add an XML element of an XML document to a partition of an array, the logic of the routine may consider the XML schema of the XML document when determining how to add the XML element to the array. For example, if the array elements of a first partition of an array are mapped to a first table, and an XML element requested to be added to the array is mapped to a second table, the logic of the routine may create a new partition, and cause the XML element to be added to the new partition. In this way, the mapping between the XML elements of the XML schema to table(s) within database 106 may be considered, by the routine, to determine whether an XML element should be added to an existing partition of an array, or instead, a new partition should be created in which the XML element should be added when client application 102 invokes the routine to add the XML element to a partition of an array.

[0039] In step 226, client application 102 determines if an amount of data maintained in the memory of client application 102 exceeds a threshold (the "memory size threshold"). If client application 102 determines that the memory of client application 102 does exceed the memory size threshold, then client application persistently stores a partition maintained in memory, e.g., client application 102 may persistently store the least recently used partition maintained in memory to database 106, although other algorithms may be used to determine which partition, maintained in memory by client application 102, should be persistently stored to database 106. In persistently storing the array elements of a single partition in database 106, client application persistently stores a portion of the XML document without persistently storing the entire XML document (unless the entire XML document can be stored within a single partition).

[0040] In step 226, if client application 102 determines that the last XML element of the XML document has been loaded into an array element of a partition in memory, then client application 102 may cause all partitions maintained in memory to be persistently stored in database 106. In this way, after the entire XML document has been processed, all partitions in memory are persistently stored to database 106 to cause any portion of the XML document, which is not currently persistently stored, to be persistently stored.

[0041] When a partition is ready to be persistently stored to database 106, client application 102 may generate a data stream from a populated partition. The data stream conforms to the format of data blocks within database 106. As a result, the data stream generated by client application 102 may be written directly to database 106 without causing the SQL engine to load each row of data, e.g., using a single batch INSERT SQL operation. Client application 102 streams the data to database server 104. A stream generated based on one array may be sent to the database server independently of a stream generated based on another array. Database server 104 writes the data received from client application 102 directly into one or more data blocks in database 106. Values in array columns that correspond to hidden columns in database tables are stored in the corresponding hidden columns as a result of the writing.

[0042] Along with the array elements, additional information maintained by the partition may be transferred to database 106, such as (a) a starting point identifier for the partition, (b) an ending point identifier for the partition, and (c) one or more set identifiers if the partition is associated with a parent array. In this way, the relationships between the XML elements of the XML documents may be maintained when the XML elements are persistently stored in database 106.

[0043] In an embodiment, each partition has its own unit of memory. All array elements of a partition use memory from the partition's memory unit. When a new partition is constructed, a new memory unit is created. When a partition is transferred to a persistent storage, such as database 106, the memory occupied by that partition can be released. This memory may be used by subsequent partitions, thereby minimizing the amount of memory required to perform operations on XML documents in memory.

[0044] In an embodiment, instead of loading the entire control file into memory, only a portion of the control file corresponding to the XML elements being currently processed by client application 102 may be loaded into memory. Client application 102 may invoke a routine to determine which portions of the control file should be loaded into memory.

IMPLEMENTING MECHANISMS

[0045] In an embodiment, client application 102, database server 104, database 106, and type implementor 108A – 108N may each be implemented on a computer system. FIG. 3 is a block diagram that illustrates a computer system 300 upon which an embodiment of the invention may be implemented. Computer system 300 includes a bus 302 or other communication mechanism for communicating information, and a processor 304 coupled with bus 302 for processing information. Computer system 300 also includes a main memory 306, such as a random access memory (RAM) or other dynamic storage device, coupled to bus 302 for storing information and instructions to be executed by processor 304. Main memory 306 also may be used for storing temporary variables or other intermediate information during execution of instructions to be executed by processor 304. Computer system 300 further includes a read only memory (ROM) 308 or other static storage device coupled to bus 302 for storing static information and instructions for processor 304. A storage device 310, such as a magnetic disk or optical disk, is provided and coupled to bus 302 for storing information and instructions.

[0046] Computer system 300 may be coupled via bus 302 to a display 312, such as a cathode ray tube (CRT), for displaying information to a computer user. An input device 314, including alphanumeric and other keys, is coupled to bus 302 for communicating information and command selections to processor 304. Another type of user input device is cursor control 316, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 304 and for controlling cursor movement on display 312. This input device typically has two degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), that allows the device to specify positions in a plane.

[0047] The invention is related to the use of computer system 300 for implementing the techniques described herein. According to one embodiment of the invention, those techniques are performed by computer system 300 in response to processor 304 executing one or more sequences of one or more instructions contained in main memory 306. Such instructions may be read into main memory 306 from another machine-readable medium, such as storage device 310. Execution of the sequences of instructions contained in main memory 306 causes processor 304 to perform the process steps described herein. In alternative embodiments, hard-wired circuitry may be used in place of or in combination with software instructions to implement the invention. Thus, embodiments of the invention are not limited to any specific combination of hardware circuitry and software.

[0048] The term "machine-readable medium" as used herein refers to any medium that participates in providing data that causes a machine to operation in a specific fashion. In an embodiment implemented using computer system 300, various machine-readable media are involved, for example, in providing instructions to processor 304 for execution. Such a medium may take many forms, including but not limited to, non-volatile media, volatile media, and transmission media. Non-volatile media includes, for example, optical or magnetic disks, such as storage device 310. Volatile media includes dynamic memory, such as main memory 306. Transmission media includes coaxial cables, copper wire and fiber optics, including the wires that comprise bus 302. Transmission media can also take the form of acoustic or light waves, such as those generated during radio-wave and infra-red data communications. All such media must be tangible to enable the instructions carried by the media to be detected by a physical mechanism that reads the instructions into a machine.

[0049] Common forms of machine-readable media include, for example, a floppy disk, a flexible disk, hard disk, magnetic tape, or any other magnetic medium, a CD-ROM, any other optical medium, punchcards, papertape, any other physical medium with patterns of holes, a RAM, a PROM, and EPROM, a FLASH-EPROM, any other memory chip or cartridge, a carrier wave as described hereinafter, or any other medium from which a computer can read.

[0050] Various forms of machine-readable media may be involved in carrying one or more sequences of one or more instructions to processor 304 for execution. For example, the instructions may initially be carried on a magnetic disk of a remote computer. The remote computer can load the instructions into its dynamic memory and send the instructions over a telephone line using a modem. A modem local to computer system 300 can receive the data on the telephone line and use an infra-red transmitter to convert the data to an infra-red signal. An infra-red detector can receive the data carried in the infra-red signal and appropriate circuitry can place the data on bus 302. Bus 302 carries the data to main memory

306, from which processor 304 retrieves and executes the instructions. The instructions received by main memory 306 may optionally be stored on storage device 310 either before or after execution by processor 304.

[0051] Computer system 300 also includes a communication interface 318 coupled to bus 302. Communication interface 318 provides a two-way data communication coupling to a network link 320 that is connected to a local network 322. For example, communication interface 318 may be an integrated services digital network (ISDN) card or a modem to provide a data communication connection to a corresponding type of telephone line. As another example, communication interface 318 may be a local area network (LAN) card to provide a data communication connection to a compatible LAN. Wireless links may also be implemented. In any such implementation, communication interface 318 sends and receives electrical, electromagnetic or optical signals that carry digital data streams representing various types of information.

[0052] Network link 320 typically provides data communication through one or more networks to other data devices. For example, network link 320 may provide a connection through local network 322 to a host computer 324 or to data equipment operated by an Internet Service Provider (ISP) 326. ISP 326 in turn provides data communication services through the world wide packet data communication network now commonly referred to as the "Internet" 328. Local network 322 and Internet 328 both use electrical, electromagnetic or optical signals that carry digital data streams. The signals through the various networks and the signals on network link 320 and through communication interface 318, which carry the digital data to and from computer system 300, are exemplary forms of carrier waves transporting the information.

[0053] Computer system 300 can send messages and receive data, including program code, through the network(s), network link 320 and communication interface 318. In the Internet example, a server 330 might transmit a requested code for an application program through Internet 328, ISP 326, local network 322 and communication interface 318.

[0054] The received code may be executed by processor 304 as it is received, and/or stored in storage device 310, or other non-volatile storage for later execution. In this manner, computer system 300 may obtain application code in the form of a carrier wave.

[0055] In the foregoing specification, embodiments of the invention have been described with reference to numerous specific details that may vary from implementation to implementation. Thus, the sole and exclusive indicator of what is the invention, and is intended by the applicants to be the invention, is the set of claims that issue from this application, in the specific form in which such claims issue, including any subsequent

correction. Any definitions expressly set forth herein for terms contained in such claims shall govern the meaning of such terms as used in the claims. Hence, no limitation, element, property, feature, advantage or attribute that is not expressly recited in a claim should limit the scope of such claim in any way. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.

2006333375 11 May 2011

The claims defining the invention are as follows:

1. A method for storing a single XML document in multiple database data blocks of a database, comprising:

at a client, storing data representing at least one XML element in a first partition of an array maintained in client memory of the client, said at least one XML element corresponding to only part of an XML document;

wherein the array is populated by array elements of a first XML type;

upon determining that a first amount of data maintained in the first partition exceeds a partition size threshold, (a) creating a new partition of the array, in the client memory, that is different and other than the first partition, and (b) subsequently loading the array elements of the first XML type into the new partition of the array;

upon determining that a second amount of data maintained in the client memory of the client exceeds a client memory size threshold, transmitting a request to a database server of said database to store data in the first partition in a database data block of said database;

transmitting a second request to said database server to store data in the new partition in a database data block of said database;

wherein transmitting a request to a database server of said database to store data in the first partition in a database data block and transmitting a request to said database server to store data in the second partition in a database data block cause the data for the first partition and data for the new partition to be stored in separate and distinct data blocks of said database; and

wherein the method is performed by one or more computing devices.

2. The method of Claim 1, further comprising determining a size of said new

partition based on, at least in part, (a) the available memory of the client or (b) a location within the database where the array elements of said new partition are to be stored.

3. The method of Claim 1, further comprising determining a size of said new partition based on, at least in part, a number of repeating XML elements within said XML document.

4. The method of Claim 1, wherein said request specifies to store a data stream that conforms to a data block format of a database managed by said database server.

5. The method of Claim 1, wherein the request to said database server to store said data in the first partition in a database is a request to persistently store second array elements of the first partition in a database using a single batch INSERT SQL operation.

6. The method of Claim 1, further comprising determining that the first partition is a least recently partition in a set of partitions maintained in the client memory of the client.

7. The method of Claim 1,

further comprising, after storing second array elements of the first partition in said database, ceasing to maintain the first partition in the client memory of the client.

8. A method for storing a single XML document in multiple database data blocks of a database, comprising:

at a client, loading one or more first array elements into a first partition of an array maintained in client memory, wherein each array element, of the one

or more first array elements, represents an XML element of an XML document;

wherein the array is populated by a plurality of array elements of a first XML type;

upon determining that a first amount of data maintained in the first partition exceeds a partition size threshold, (a) creating a new partition of the array, in the client memory, that is different and other than the first partition, and (b) subsequently loading the plurality of array elements of the first XML type into the new partition of the array;

upon determining that a second amount of data maintained in the client memory exceeds a client memory size threshold, persistently storing the first array elements, of the first partition, in said database without persistently storing the entire XML document;

transmitting a request to a database server of said database to store data in the new partition in a database data block of said database;

wherein transmitting a request to a database server to store data in the first partition in a database data block and persistently storing the first array elements, of the first partition, in said database without persistently storing the entire XML document cause the data for the first partition and data for the new partition to be stored in separate and distinct data blocks of said database; and

wherein the method is performed by one or more computing devices.

9. A computing device comprising one or more processors and a volatile or non-volatile machine-readable medium storing instructions, wherein the instructions, when executed by the one or more processors, cause the one or more processors to perform the method of any one of Claims 1 through 8.

10. A volatile or non-volatile machine-readable medium carrying instructions, wherein the instructions, when executed by one or more processors, cause the one or more processors to perform the steps of any one of Claims 1 through 8.

11. A method for storing a single XML document in multiple database data blocks of a database, substantially as herein described with reference to the accompanying drawings.

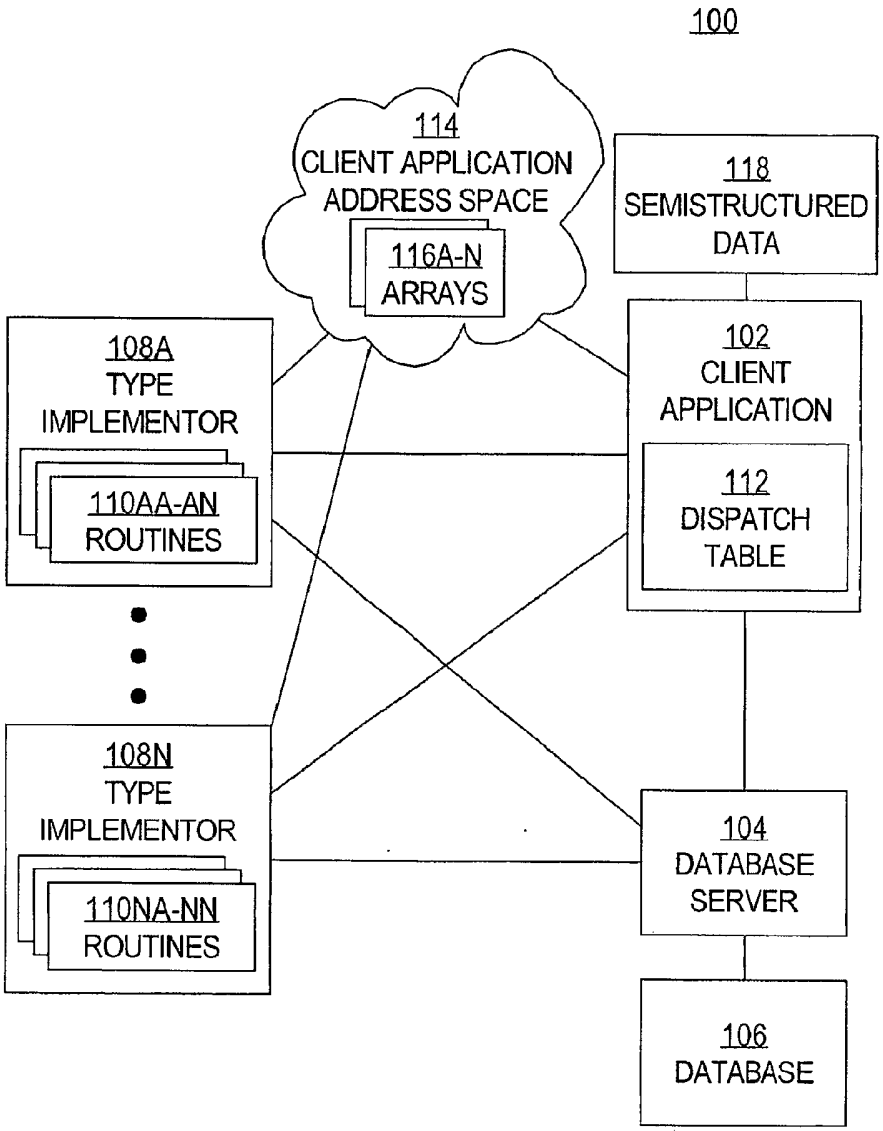

100
114
CLIENT APPLICATION ADDRESS SPACE
116A-N ARRAYS
118
SEMISTRUCTURED DATA
108A
TYPE
IMPLEMENTOR
110AA-AN
ROUTINES
102
CLIENT
APPLICATION
112
DISPATCH
TABLE
108N
TYPE
IMPLEMENTOR
110NA-NN
ROUTINES
104
DATABASE
SERVER
106
DATABASE

FIG. 1

FIG. 2

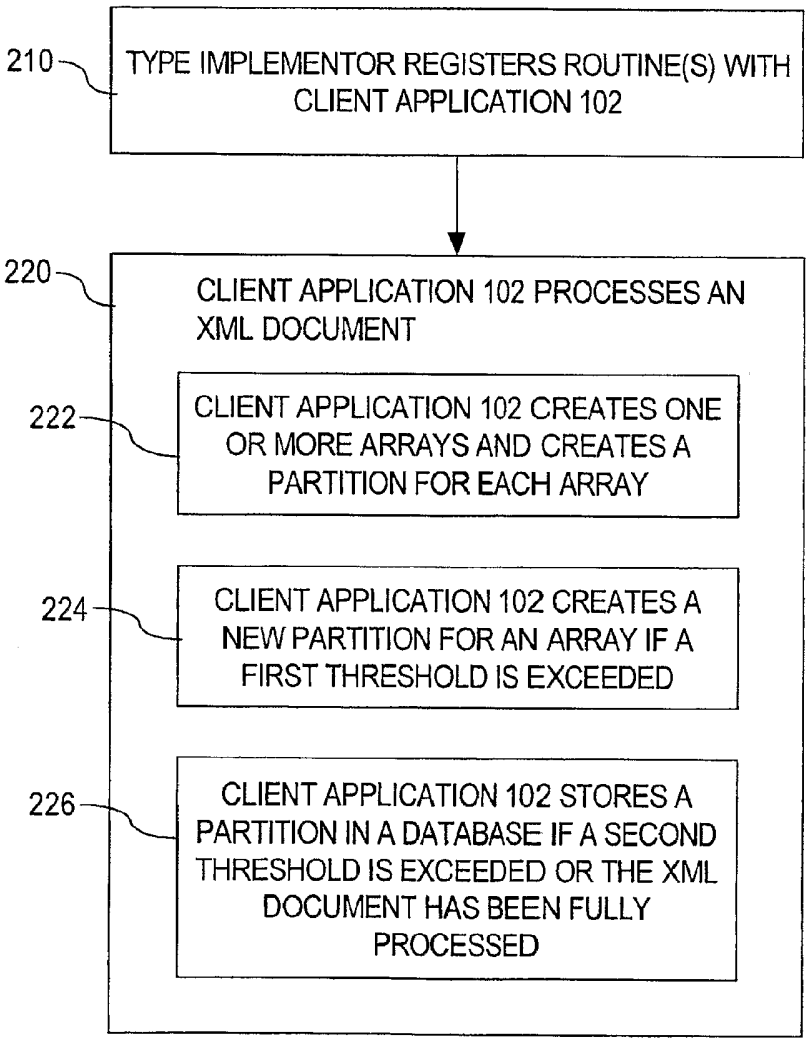
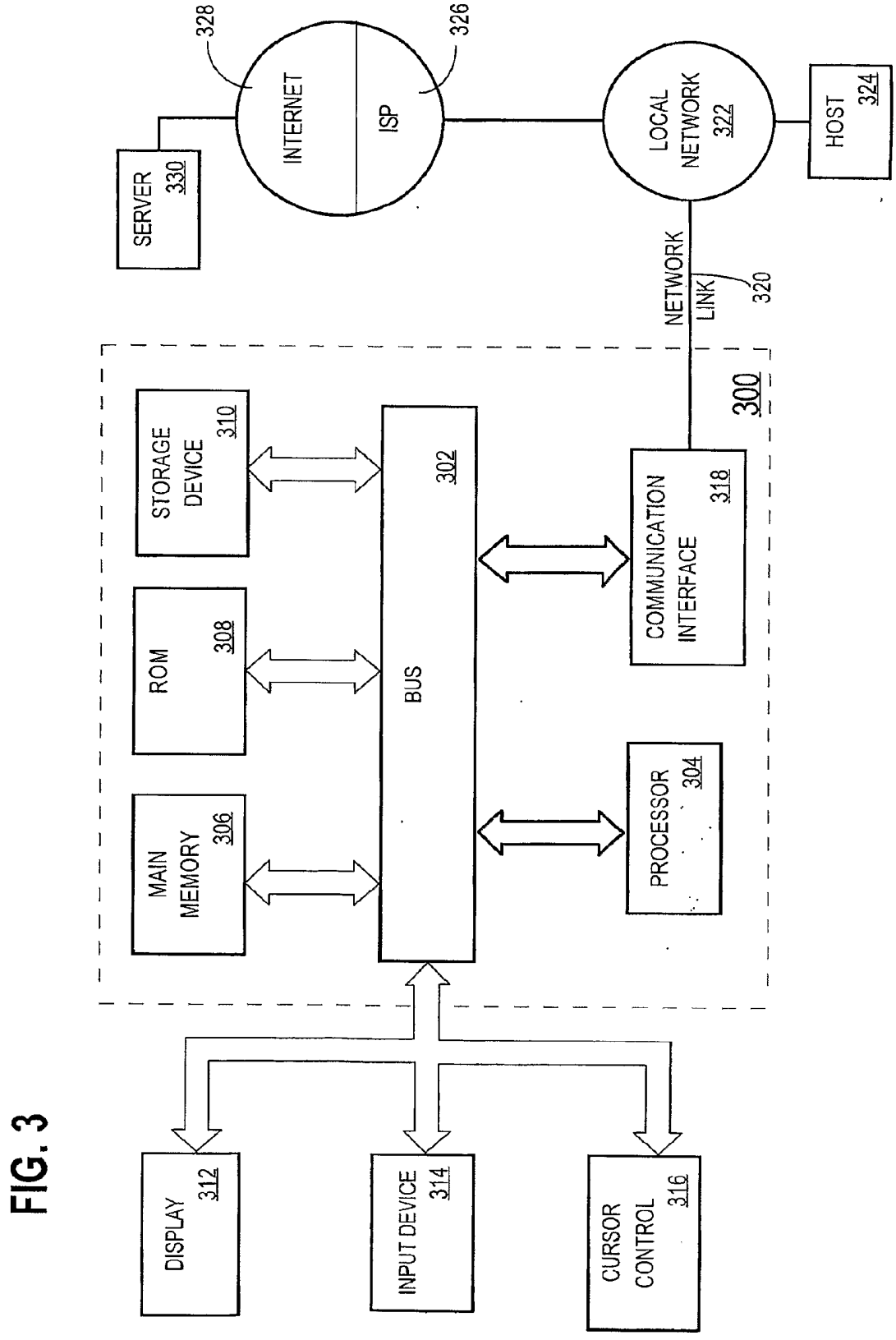

210
TYPE IMPLEMENTOR REGISTERS ROUTINE(S) WITH CLIENT APPLICATION 102
220
CLIENT APPLICATION 102 PROCESSES AN XML DOCUMENT
222
CLIENT APPLICATION 102 CREATES ONE OR MORE ARRAYS AND CREATES A PARTITION FOR EACH ARRAY
224
CLIENT APPLICATION 102 CREATES A NEW PARTITION FOR AN ARRAY IF A FIRST THRESHOLD IS EXCEEDED
226
CLIENT APPLICATION 102 STORES A PARTITION IN A DATABASE IF A SECOND THRESHOLD IS EXCEEDED OR THE XML DOCUMENT HAS BEEN FULLY PROCESSED

FIG. 3


328
INTERNET
ISP
326
SERVER 330
LOCAL NETWORK 322
HOST 324
NETWORK LINK 320
300
STORAGE DEVICE 310
ROM 308
MAIN MEMORY 306
BUS 302
COMMUNICATION INTERFACE 318
PROCESSOR 304
DISPLAY 312
INPUT DEVICE 314
CURSOR CONTROL 316