OPIC
Office de la propriété intellectuelle du Canada

CIPO
Canadian Intellectual Property Office

(12) (19) (CA) **Demande-Application**

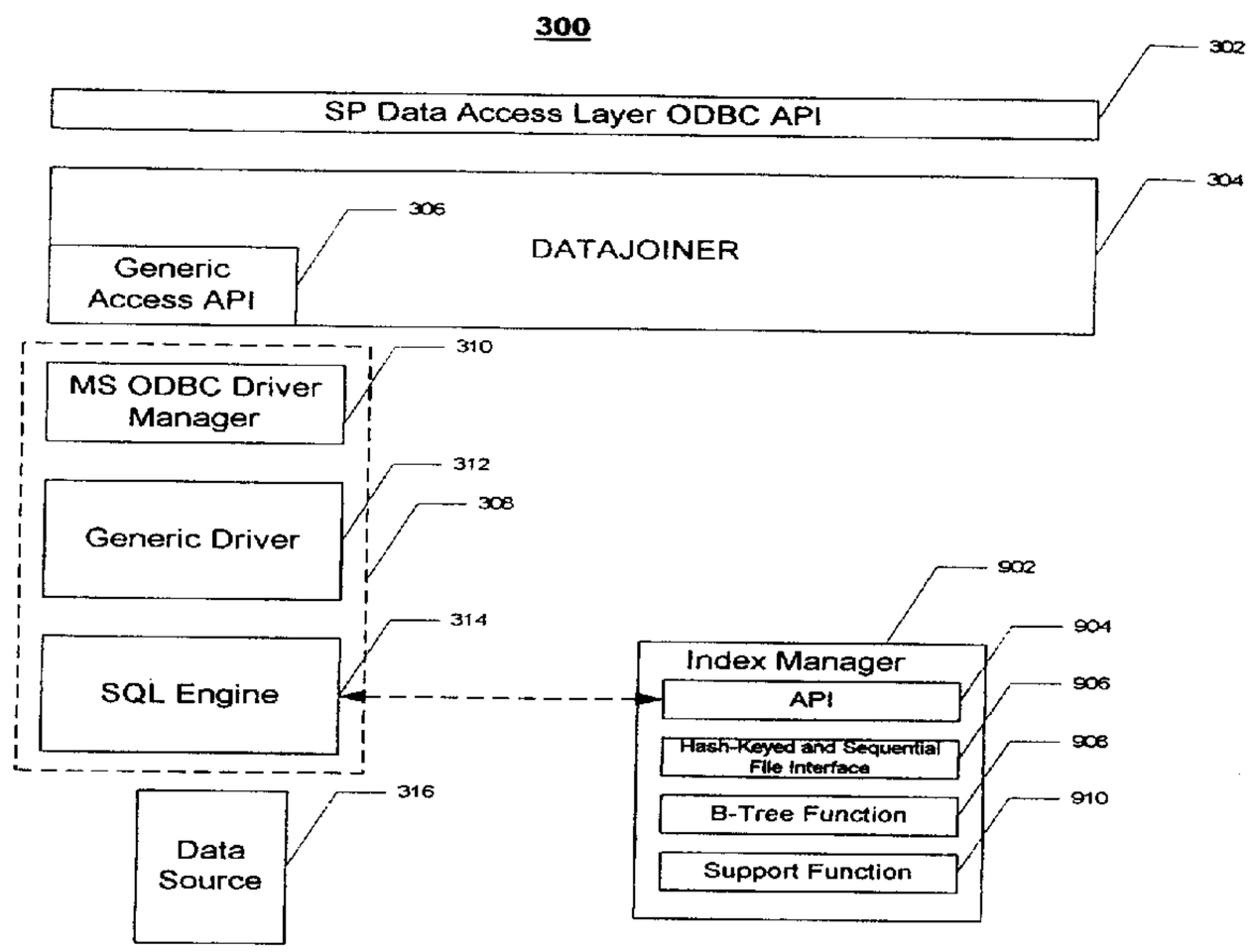
(21) (A1) **2,305,122**
(22) 2000/04/12
(43) 2001/01/08

(72) GAJDA, KIMBERLY L., US
(72) HESS, BRADLEY A., US
(72) KEMP, JULIE A.H., US
(72) LEWIS, JERRY J., US
(72) THORPE, ROBERT T., US
(71) INTERNATIONAL BUSINESS MACHINES CORPORATION, US

(51) Int.Cl.[7] G06F 17/30, H04L 12/16
(30) 1999/07/08 (09/349,607) US

(54) **METHODE ET SYSTEME D'ACCES AMELIORE A DES BASES DE DONNEES NON RELATIONNELLES**

(54) **METHOD AND SYSTEM FOR IMPROVED ACCESS TO NON-RELATIONAL DATABASES**



(57) The method and system of the present invention provides improved access to databases in a computer system. The present invention includes a plurality of data sources, the data sources including at least one non-relational data source; a server coupled to the plurality of data sources, the server capable of providing client access to the plurality of data sources; and a store place data access layer (SPDAL) coupled to the server and the at least one non-relational data source, the SPDAL providing relational features to the at least one non-relational data source. In the preferred embodiment, the relational features provided include a two-phase commit process; a process for performing Structured Query Language (SQL) operations from within a single application; a referential integrity process; and a process for providing indices for non- relational data source files. These features allow the computer system to perform data changes across multiple database sources within a single operation; perform SQL operations from a single application across heterogeneous file systems; provide referential integrity of data in both relational and non- relational databases; and perform ad-hoc queries of data and other queries efficiently in non-relational databases. By adding these features, significant flexibility is added to the computer system.

METHOD AND SYSTEM FOR IMPROVED ACCESS TO NON-RELATIONAL DATABASES

ABSTRACT

The method and system of the present invention provides improved access to databases in a computer system. The present invention includes a plurality of data sources, the data sources including at least one non-relational data source; a server coupled to the plurality of data sources, the server capable of providing client access to the plurality of data sources; and a store place data access layer (SPDAL) coupled to the server and the at least one non-relational data source, the SPDAL providing relational features to the at least one non-relational data source. In the preferred embodiment, the relational features provided include a two-phase commit process; a process for performing Structured Query Language (SQL) operations from within a single application; a referential integrity process; and a process for providing indices for non-relational data source files. These features allow the computer system to perform data changes across multiple database sources within a single operation; perform SQL operations from a single application across heterogeneous file systems; provide referential integrity of data in both relational and non-relational databases; and perform ad-hoc queries of data and other queries efficiently in non-relational databases. By adding these features, significant flexibility is added to the computer system.

METHOD AND SYSTEM FOR IMPROVED ACCESS TO NON-RELATIONAL DATABASES

FIELD OF THE INVENTION

The present invention relates to computer systems, and more particularly to the access of databases in the computer system.

BACKGROUND OF THE INVENTION

Databases are an important tool for the storage and management of information for businesses. Both relational and non-relational databases exist for this purpose. Examples of relational databases include ORACLE, DB2, AND INFORMIX. Examples of non-relational databases include custom databases created with the 4690 operating system, developed by INTERNATIONAL BUSINESS MACHINES CORPORATION. The 4690 operating system allows programmers to create custom databases which support keyed, sequential, and binary file types. Database Management Systems (DBMS) provide users the capabilities of controlling read/write access, specifying report generation, and analyzing usage.

Some businesses, particularly large organizations, maintain multiple database types, both relational and non-relational. The simultaneous access to these databases then becomes an issue. For example, a company may wish to combine and compare customer preference information in the marketing department's ORACLE database with the current customer issues tracked in the customer service department's DB2 database. One conventional way of accessing these databases is through the DB2 DATAJOINER ("DataJoiner") products, developed by INTERNATIONAL BUSINESS MACHINES CORPORATION. DataJoiner is a multi-database server which uses the Structured Query Language (SQL) and Open Database Connectivity (ODBC), to provide client access to diverse data sources that reside on different platforms.

Figure 1 illustrates a computer system which uses DataJoiner. The computer system 100 comprises clients 102 in various platforms, a DataJoiner server 104, and data sources 106 of various

types. The Data Joiner server 104 allows the clients 102 to transparently access data from the different data sources 106.

Figure 2 is a block diagram illustrating the basic architecture of DataJoiner. DataJoiner 202 is a DBMS which includes ODBC software and drivers for a variety of data sources. DataJoiner 202 allows for transparent access to multiple data sources in a heterogeneous environment. The Generic Access Applications Programming Interface (API) 204 allows for customized access to home-grown or other data sources 208 not already supported by DataJoiner. Generic Data Access Modules 206 containing Custom ODBC compliant drivers for these data sources may be written and installed on the same system as DataJoiner 202 and used by DataJoiner 202 to access the data sources 208. The Generic Access API 204 defines the calls which must be provided by the custom ODBC drivers in order for DataJoiner 202 to access the custom data source 208.

However, DataJoiner is not able to provide certain features for access to the non-relational databases. In particular, DataJoiner 202 does not provide (1) a two-phase commit, (2) performance of SQL operations from within a single application across heterogeneous file systems, (3) referential integrity, and (4) indices for queries across heterogeneous file systems which include non-relational data sources, such as 4690 keyed, sequential, and binary file systems.

Two-Phase Commit

DataJoiner can perform data changes across multiple databases sources within a single operation, while guaranteeing that changes to one source can be rolled back if changes to another source within the same transaction fail. The two-phase commit is the mechanism for implementing this. For example, someone may be trying to wire some money electronically to another location. A debit would need to be written against one account maintained by a first DBMS, and a credit would have to be reflected in another account maintained by a second DBMS. The two-phase commit protocol ensures that either all of the DBMS involved commit the changes required by the transaction or none do. However, keyed, sequential, and binary file systems cannot participate in this function.

SQL Operations From Within a Single Application

Conventional relational DBMS's employ two underlying data storage concepts in order to facilitate relational operations on that data: (1) data is stored in tables with each table having a fixed number of columns; and (2) system catalogs, or data dictionaries, are generated for each table. These data dictionaries (metadata) describe the column layout, index structure, and other data access information. These concepts allow one application to create data, and other applications totally unrelated to the first, to analyze and update the data.

However, with keyed, sequential, and binary file systems, there are several inhibitors that prevent these file systems from having the above flexibility. These include: (1) data is not held in a fixed table format; (2) data access is via a buffer that passes the entire record, with no public or obvious field delineation available; and (3) the structure, or field layout, of keyed, sequential, and binary records is not stored in a publicly available catalog. With these inhibitors, relational operations cannot be performed on the keyed, sequential, and binary file systems, making SQL operations from within a single application across heterogeneous file systems which include these file systems difficult or impossible.

Referential Integrity

Referential integrity is a technique where the consistency of data within a DBMS is ensured by referring to one piece of data during an operation on another piece of data. An example would be to refer to a department table to ensure that a valid department number is being used when a new employee is being added to the database. Referential integrity is incorporated in many of the conventional relational DBMS and is being enforced at the database level to ensure that all applications adhere to any constraints and thus ensure the integrity of the data.

However, the concept of referential integrity, and a file system mechanism to enforce it, does not exist for keyed, sequential, and binary file systems. Without this function, heterogeneous DBMS can not implement referential integrity between the relational and non-relational data. Using the above example, if the valid department list is in a keyed file system and the new employee data was

to be stored in a relational database, there would be no way that the underlying data stores could enforce the referential integrity during the employee add operation.

Indices for Queries

A need often arises to access keyed and sequential data based on data selection criteria different from the key or record number traditionally used to access the data. Queries based on this type of selection criteria are referred to as ad-hoc queries. For instance, instead of retrieving an employee record by an employee number there may be a need to retrieve all of the employees who are employed in department "X". The employee records are keyed by an employee identification number yet the selection criteria is based on the department number in which the employee is employed.

For relational databases, ad-hoc queries may be efficiently satisfied since indices of the data may be created for data elements other than the key or record number. Once created, these indices may be used as an alternative form of data access which provides a more efficient processing of the query.

However, to satisfy queries using conventional file access mechanisms available for hash-keyed or sequential files, every record in the file must be read and compared to the selection criteria since indices are not available for these files. This causes performance problems both in accessing the file and in creating the network traffic when the querying application resides on a machine other than the source data files.

Accordingly, there exists a need for a method and system for improved access to non-relational databases in a computer system. The method and system should increase the flexibility and efficiency of access to the databases. The present invention addresses such a need.

SUMMARY OF THE INVENTION

The method and system of the present invention provides improved access to databases in a computer system. The present invention includes a plurality of data sources, the data sources including at least one non-relational data source; a server coupled to the plurality of data sources, the server capable of providing client access to the plurality of data sources; and a store place data access layer (SPDAL) coupled to the server and the at least one non-relational data source, the SPDAL providing relational features to the at least one non-relational data source. In the preferred embodiment, the relational features provided include a two-phase commit process; a process for performing Structured Query Language (SQL) operations from within a single application; a referential integrity process; and a process for providing indices for non-relational data source files. These features allow the computer system to perform data changes across multiple database sources within a single operation; perform SQL operations from a single application across heterogeneous file systems; provide referential integrity of data in both relational and non-relational databases; and perform ad-hoc queries of data and other queries efficiently in non-relational databases. By adding these features, significant flexibility is added to the computer system.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a diagram illustrating a computer system which uses DataJoiner.

Figure 2 is a block diagram illustrating the basic architecture of DataJoiner.

Figure 3 is a block diagram illustrating a preferred embodiment of the architecture of the method and system in accordance with the present invention.

Figure 4 is a block diagram illustrating the X/OPEN DTP model used in accordance with the present invention.

Figure 5 is a flow chart illustrating the two-phase commit process provided by the method and system in accordance with the present invention.

Figure 6 is a block diagram illustrating an example of a two-phase commit performed in accordance with the present invention.

Figure 7 is a preferred embodiment of metadata tables provided for non-relational databases in accordance with the present invention.

Figure 8 is a flow chart illustrating the referential integrity process provided by the method and system in accordance with the present invention.

Figure 9 is a block diagram illustrating a preferred embodiment of the Index Manager in accordance with the present invention.

Figure 10 is a flow chart illustrating the creation of indices by the Index Manager in accordance with the present invention.

Figure 11 is a flow chart illustrating the use of the indices in satisfying a user query in accordance with the present invention.

DETAILED DESCRIPTION OF THE INVENTION

The present invention relates to a method and system for providing improved access to non-relational databases in a computer system. The following description is presented to enable one of ordinary skill in the art to make and use the invention and is provided in the context of a patent application and its requirements. Various modifications to the preferred embodiment will be readily apparent to those skilled in the art and the generic principles herein may be applied to other embodiments. Thus, the present invention is not intended to be limited to the embodiment shown but is to be accorded the widest scope consistent with the principles and features described herein.

The present invention provides improved access to non-relational databases through an expansion upon the capabilities of DataJoiner. Figure 3 is a block diagram illustrating a preferred embodiment of the architecture of the method and system of the present invention. The Store Place Data Access Layer (SPDAL) architecture 300 comprises an Open Database Connectivity Applications Programming Interface 302 (ODBC API) which functions with DataJoiner 304. The SPDAL ODBC API 302 works with the Generic Access API 306 of DataJoiner 304 to provide a single access point for heterogeneous data source manipulations. An ODBC Driver 308 may be created to function with the SPDAL ODBC API 302 to access keyed, sequential, and binary file types, such as a 4690 data source 316. The ODBC Driver 308 comprises an ODBC Driver Manager 310, a Generic Driver 312, and a SQL Engine 314. In the preferred embodiment, the ODBC Driver Manager 310 is an ODBC Driver Manager, known in the art.

SPDAL 300 improves access to non-relational databases by providing the following capabilities: (1) true two-phase commit, (2) performance of SQL operations from within a single application across heterogeneous file systems, (3) referential integrity, and (4) indices for queries.

Two-Phase Commit

A two-phase commit process for keyed, sequential, and binary file systems is provided by the preferred embodiment of the method and system of the present invention. The two-phase commit is provided with an X/OPEN Distributed Transaction Processing (DTP) model using "XA interface" in SPDAL 302. X/OPEN, and its XA interface, is a standard protocol for access to databases.

Figure 4 is a block diagram illustrating the X/OPEN DTP model used in accordance with the present invention. The X/OPEN DTP model comprises three software components:

- An application program 402 which defines transaction boundaries and specifies actions that constitute a transaction.
- Resource managers 404 which provide access to shared resources.
- A transaction manager 406 which assigns identifiers to transactions, monitors their progress and takes responsibility for transaction completion and for failure recovery.

In the preferred embodiment, the XA Interface 408 is implemented with the X/OPEN DTP model as an intercept layer in the Generic Driver 312. The intercept layer would act as a resource manager for each file system, i.e., it would assume the role of a data source. DataJoiner 304 would act as the transaction manager. This would make the keyed, sequential, or binary file system appear to DataJoiner 304 as a fully functional relational database implementing an XA interface 408. The XA Interface 408 implements XA calls which allow for interaction between the transaction manager/DataJoiner 304 and the data source 316. This two-phase commit protocol provided by the present invention can be used to manage data changes across multiple data base sources within a single operation.

Figure 5 is a flow chart illustrating the two-phase commit process provided by the method and system in accordance with the present invention. For this illustration, assume that a first

database is a non-relational database and a second database is a relational database and a request requires a two-phase commit between them. First, DataJoiner 304, as the transaction manager, sends a request to the non-relational database, via step 510, through the intercept layer in the Generic Driver 312. The intercept layer implements the appropriate XA calls to facilitate interaction between DataJoiner 304 and the non-relational database. The non-relational database replies to the request and waits, via step 520. DataJoiner 304 also sends a request to the relational database, via step 530. The relational database replies to the request and waits, via step 540. If both the non-relational and relational databases commit to the request, via step 550, the request is performed, via step 560. Otherwise, the transaction fails, via step 570.

Examples of functions for each XA API call comprise the below:

- **xa_close**: Frees any dedicated two-phase commit resources.
- **xa_commit**: Unlock the section of the file previously locked, this will make the data changes available to other applications.
- **xa_complete**: Used by the transaction manager to wait for asynchronous operations. Return control to the transaction manager when the operation identified by the handle is complete.
- **xa_end**: Free any dedicated resources acquired during xa_start processing.
- **xa_forget**: Release resources associated with a xid.
- **xa_open**: Initialize any dedicated two-phase commit resources.
- **xa_prepare**: Obtain a write lock on the portion of the keyed, sequential, or binary file that will be changed. By acquiring a write lock, the resource manager intercept layer can guarantee that the subsequent write and commit will occur. The prepared write instruction (location in the file and buffer to write) should be saved in case a rollback is required.
- **xa_recover**: Return to the transaction manager any prepared but not committed transactions.
- **xa_rollback**: Undo the write operation staged with xa_prepare.
- **xa_start**: Initialize any dedicated resources required for a transaction.

Figure 6 illustrates an example of a two-phase commit performed in accordance with the present invention. In this example, assume that a request is received by DataJoiner to transfer $1000 from a first account at a first bank to a second account in a second bank. The first bank maintains its accounts in a non-relational database 602 (DS1), such as a 4690 data source. The second bank maintains its accounts in a relational database 604 (DS2), such as a DB2 data source. The banks do not want to perform the credit without also performing the debit and vice versa. In completing this transaction, first the transaction manager, DataJoiner 304, sends a request to the 4690 Keyed File, DS1 602, to debit $1000 from the first account, via step 510. The request is sent through the Generic Access API 306 to the Intercept Layer in the Generic Driver 312. The Intercept Layer implements the appropriate XA calls to facilitate the portion of the transaction involving DS1 602. DS1 602 replies to the XA calls and waits, via step 520. DS1's 602 reply is sent to the transaction manager by the Intercept Layer. The transaction manager also sends a request to DS2 604 to credit the second account $1000, via step 530. DS2 604 replies and waits, via step 540. This ends phase one of the two-phase commit process.

If either DS1 602 or DS2 604 indicates that they are unable to make the requested credit/debit, i.e., if either data source fails to commit, via step 550, the transaction fails, via step 570, and the transaction manager moves on to the next transaction. If both DS1 602 and DS2 604 indicate they are able to make the requested credit/debit, i.e., both data sources commit, via step 550, then the credit and debit are performed, via step 560. This ends phase two of the two-phase commit process.

In the above manner, a two-phase commit process is provided by the method and system of the present invention.

SQL Operations From Within a Single Application

With the preferred embodiment of the method and system of the present invention, the ability to perform SQL operations from within a single application is provided. The present invention implements this capability by providing additional metadata tables in a database in addition to the metadata tables of DataJoiner 304. These additional metadata tables describe the keyed, sequential, and binary file content such that the files within those file systems can be accessed by the SPDAL ODBC API 302 of the present invention, and manipulate it using relational operations. In addition, the metadata content itself is described and held as a relational table, so that this data is publicly available to all users and applications, and can be used in conjunction with DataJoiner 304 to perform single relational operations across relational data sources and keyed, sequential, and binary file data concurrently.

The metadata information of the present invention is used by DataJoiner 304 via the SPDAL ODBC API 302 to service requests from applications, thus removing the burden of individual applications to code to the specifics of the data's physical design and location.

A summary of the metadata tables of the present invention follows. A more detailed description of fields in the metadata tables are provided in Figure 7:

- **LogicalEntity Table**: maps a LogicalEntityName to one or more physical entities. (See Table 1, Figure 7)
- **PhysicalEntity Table**: describes metadata information that is defined for each relational database table or keyed, sequential, and binary file. (See Table 2, Figure 7)
- **RecordDescription Table**: describes the metadata information for records that exist within keyed, sequential, or binary files. (See Table 3, Figure 7)
- **Attribute Table**: describes the metadata information for all fields in a keyed, sequential, binary file. The definition of a field provides the physical layout structure that an application would use to process the data within the field. (See Table 4, Figure 7)
- **ValidAttributeValuesTable**: describes the discrete values that are valid within an attribute. (See Table 5, Figure 7)
- **ForeignKey Table**: describes the metadata information for parent-child relationships. These relationships are used to enforce referential integrity across all SPDAL data, as will be described

in more detail later in this specification. This includes relational databases as well as non-relational databases. Relationships can be established between attributes within the same file system (for example between two relational tables) or between attributes across different file systems (for example between a relational table and a keyed file). (See Table 6, Figure 7)

- **Index Table**: describe, set, and maintain indexes on keyed, sequential, binary files. The use of this table will be described in more detail later in this specification. (See Table 7, Figure 7)

Although the present invention is described with the metadata tables as illustrated in Figure 7, one of ordinary skill in the art will understand that other access mechanisms for keyed, binary, and sequential file types may be used without departing from the spirit and scope of the present invention.

When DataJoiner 304 receives a SQL query which involves both a relational and a non-relational data source, DataJoiner 304 "splits" the query. It passes "as is" to the SPDAL ODBC API 302 the portion of the query which deals with the non-relational data source 316, and it passes the other portion to the relational data source through its native ODBC driver. The SQL engine 314 then interprets the SQL query into the low level calls for access to the non-relational data source 316. The requested data is returned in response to these low level calls. This result is then returned to DataJoiner 304, which combines it with the result returned from the relational data source. DataJoiner 304 then returns the combined result to the SQL query.

By providing the metadata in accordance with the present invention, advanced relational concepts, such as referential integrity, can be used across relational database management systems, and keyed, sequential, and binary file systems in the same operation. In addition, the metadata content structure is described as standard relational tables, making the metadata publicly available to any user or application. Finally, the metadata allows for a logical view for access to the underlying keyed or sequential file data. This enables an application to code to a logical model without having to be concerned with the underlying physical data store, or changes to the underlying physical data store.

Referential Integrity

The method and system of the present invention provides for referential integrity across both relational and non-relational databases. In the preferred embodiment, referential integrity is implemented in the SPDAL ODBC API 302 by combining the metadata described above with a relational database engine that can process the results of relational operations from outside data sources to ensure referential integrity across both the relational database and keyed, sequential, and binary file systems. The relational database engine is a SQL engine 314 which has the ability to perform queries which implement referential integrity, as described below.

The metadata is used to describe the referential constraints within the keyed, sequential, and binary file systems, and between these file systems and the relational file systems. The metadata is implemented by providing the following information in its ForeignKey Table:

- Parent/child relationships between files
- Primary key fields
- Cardinality between the parent and child
- List of fields that comprise a foreign key
- Delete rules

Table 6 in Figure 7 provides in more detail the above listed information.

Figure 8 is a flowchart illustrating the referential integrity process provided by the method and system of the present invention. In the preferred embodiment, the database access is through ODBC or X/OPEN CLI interfaces. Both are widely accepted and implemented by conventional competitive relational database management systems. First, the SPDAL ODBC API 302 captures an end user query before it is passed to DataJoiner 304, via step 810. The query is preprocessed in the SPDAL ODBC API 302 based on information in the metadata tables, via step 820, specifically, the information in the ForeignKey Table. In the preprocessing, the SPDAL ODBC API 302 determines if the end user query requires any referential integrity processing, via step 822. Is so, the SPDAL ODBC API 302 would formulate the queries to execute and enforce the referential constraints, via step 824. Then, the end user query is passed to DataJoiner 304 for processing, via step 840. DataJoiner 304 determines if the query requires non-relational data, via step 850. If not, DataJoiner processes the query for the relational data, via step 860, and returns the result, via step

870. If so, then the query is processed on the non-relational data source, via step 880. In this processing, the queries which execute and enforce the referential constraint is passed to the SQL Engine 314, via step 826. The SQL Engine 314 translates these queries to commands compatible with the non-relational data source 316, via step 828. The SQL Engine 314 next executes these commands, via step 830. DataJoiner also processes the query for the relational data, via step 860. The results from both the relational and non-relational data sources are returned, via step 870.

The following is a summary of ODBC calls which may be made by the ODBC API 302 to implement referential constraints in accordance with the method and system of the present invention:

- DELETE: Use the delete rules specified in the metadata to delete restrict, set null, or cascade. The ODBC driver of the present invention may need to make several calls into the relational DBMS in order to complete the requested action.
- INSERT: Issue a SELECT ODBC call on the parent table to ensure that each row conforms to the referential constraint prior to passing the INSERT to the original relational DBMS ODBC driver. If the constraint does not pass, then the ODBC driver of the present invention will generate an error message and return control directly to the application program. From the application's viewpoint, it looks as if the underlying relational DBMS detected the referential constraint violation and returned the error.
- UPDATE: Issue a SELECT ODBC call on the parent table to ensure that each row conforms to the referential constraint prior to passing the UPDATE to the original relational DBMS ODBC driver. If the constraint does not pass, then the ODBC driver of the present invention will generate an error message and return control directly to the application program. From the application's viewpoint, it looks as if the underlying relational DBMS detected the referential constraint violation and returned the error.

In this manner, referential integrity is provided within a keyed, sequential, or binary file system in accordance with the method and system of the present invention. The present invention provides dynamic real-time enforcement of the referential integrity constraints, at the data store level, across both relational databases and keyed, sequential, and binary file systems.

Indices for Queries

The method and system of the present invention provides a more efficient performance of queries, including ad-hoc queries, for non-relational databases. The efficiency is increased through the creation of an Index Manager which allows for the creation of indices for hash-keyed and sequential files. The indices are created based on the data selection criteria commonly used to satisfy user-defined ad-hoc queries. The user decides, based on the frequency that the query will be issued, along with the required response time to obtain the result set, which indices should be created.

As illustrated in Figure 9, the Index Manager 902 is located next to the SQL Engine 314. The Index Manager 902 is accessed by the SQL Engine 314 during the satisfying of an end user query. In the preferred embodiment, the Index Manager comprises the following:

- **Index Manager API 904**: allows users to maintain their indices and access the data contained within the indices. This interface is available for use by external application programs that require access to the indices.
- **Index Manager Hash-keyed and Sequential File Interface 906**: provides a means of updating indices to reflect additions, updates, and deletions of data from indices when the corresponding source data files change. This interface is context sensitive in that it depends on the file maintenance mechanism available on the data source.
- **Index Manager B-Tree Function 908**: index entries are maintained in a B-Tree structure within the Index Manager to maintain the index entries. B-Tree structures are well known in the art and will not be discussed further here.
- **Index Manager Support Functions 910**: include error logging and audit logging facilities, as well as a means of obtaining information about the data being indexed, i.e., the meta data.

Figure 10 is a flow chart illustrating the creation of indices by the Index Manager 802 in accordance with the present invention. First, hash-keyed and/or sequential files are created, via step 1010. Next, the Index Manager API 904 is used to specify an index or indices to be created on a field or fields within a file, via step 1020. The Index Manager 902 then creates an index on the

specified field or fields, via step 1030. This index is kept up to date when changes are occurring in the source data files via the Index Manager Hash-keyed and Sequential File Interface 906.

Figure 11 is a flowchart illustrating the use of the indices in satisfying a user query in accordance with the present invention. When the user executes a query, via step 1110, the query is sent to the SQL Engine 314, via step 1120. The SQL Engine 314 then accesses the Index Manager 902 via the Index Manager API 904 to see whether an index exists that helps satisfy the query, via step 1130. If so, the Index Manager API 904 makes function calls to retrieve the indexed data, via step 1140. If not, then every record in the file needs to be read and compared to the selection criteria, via step 1150. Once the index data is retrieved, the Index Manager 902 determines if the index alone satisfies the query, via step 1160. If it does, then the result of the query is returned to DataJoiner 304 via the SQL Engine 314, via step 1170. If not, the rows based on the keyed file keys or binary sequential file offsets found in the index are also retrieved via the SQL Engine 314, via step 1180. Then the result is returned, via step 1170.

With the Index Manager 902 of the present invention, indexing capabilities are provided to hash-keyed, sequential and binary files. The efficiency in the execution of other queries is also greatly improved. By allowing the query application processing to only retrieve data that is known based on indices to satisfy the user defined query, there will be less network traffic when the application issuing the query resides on a machine other than the one where the source data files are stored. In addition, the cost of disk space to store indices is lowered.

A method and system for improved access to non-relational databases in a computer system has been disclosed. The method and system provides capabilities for a two-phase commit, performance of SQL operations from within a single application across heterogeneous file systems, referential integrity, and indices for ad-hoc and other queries for non-relational databases. These capabilities allow the computer system to perform data changes across multiple database sources within a single operation; perform SQL operations from a single application across heterogeneous file systems; provide referential integrity of data in both relational and non-relational databases; and perform ad-hoc queries of data and other queries efficiently when some or all of the data is stored in non-relational databases. By adding these capabilities, significant flexibility is added to the computer system.

Although the present invention has been described in accordance with the embodiments shown, one of ordinary skill in the art will readily recognize that there could be variations to the embodiments and those variations would be within the spirit and scope of the present invention. Accordingly, many modifications may be made by one of ordinary skill in the art without departing from the spirit and scope of the appended claims.

The embodiments of the invention in which an exclusive property or privilege is claimed are defined as follows:

1. A system for improved access to databases in a computer system, comprising:

a plurality of data sources, the data sources including at least one non-relational data source;

a server coupled to the plurality of data sources, the server capable of providing client access to the plurality of data sources; and

a store place data access layer (SPDAL) coupled to the server and the at least one non-relational data source, the SPDAL providing relational features to the at least one non-relational data source.

2. The system of claim 1, wherein the relational features comprise:

a two-phase commit process;

a process for performing Structured Query Language (SQL) operations from within a single application;

a referential integrity process; and

a process for providing indices for non-relational data source files.

3. The system of claim 1, wherein the server is a DB2 DataJoiner.

4. The system of claim 3, wherein the DB2 DataJoiner comprises a Generic Access Applications Programming Interface (API) which allows for customized access to data sources not already supported by DataJoiner.

5. The system of claim 1, wherein the SPDAL comprises:

a SPDAL Open Database Connectivity (ODBC) API coupled to the server;

an ODBC Driver Manager coupled to a Generic Access API;

a Generic Driver coupled to the ODBC Driver Manager; and

a SQL Engine coupled to the Generic Driver.

6. The system of claim 5, wherein the Generic Driver comprises an intercept layer, the intercept layer functioning as a resource manager for the at least one non-relational database during a two-phase commit process.

7. The system of claim 5, further comprising an index manager coupled to the SQL Engine, the index manager allowing for the creation of indices for the at least one non-relational data source.

8. The system of claim 7, wherein the index manager comprises:

an index manager API;

a hash-keyed and sequential file interface;

a B-Tree function; and

a support function.

9. A system for improved access to databases in a computer system, comprising:

a plurality of data sources, the data sources including at least one non-relational data source;

a server coupled to the plurality of data sources, the server capable of providing client access to the plurality of data sources; and

a SPDAL coupled to the server and the at least one non-relational data source, the SPDAL providing relational features to the at least one non-relational data source, wherein the relational features comprises:

a two-phase commit process;

a process for performing SQL operations from within a single application;

a referential integrity process; and

a process for providing indices for non-relational data source files.

10. The system of claim 9, wherein the server is a DB2 DataJoiner.

11. The system of claim 10, wherein the DB2 DataJoiner comprises a Generic Access API which allows for customized access to data sources not already supported by DataJoiner.

12. The system of claim 9, wherein the SPDAL comprises:

a SPDAL ODBC API coupled to the server;

an ODBC Driver Manager coupled to a Generic Access API;

a Generic Driver coupled to the ODBC Driver Manager; and

a SQL Engine coupled to the Generic Driver.

13. The system of claim 12, wherein the Generic Driver comprises an intercept layer, the intercept layer functioning as a resource manager for the at least one non-relational database during a two-phase commit process.

14. The system of claim 12, further comprising an index manager coupled to the SQL Engine, the index manager allowing for the creation of indices for the at least one non-relational data source.

15. The system of claim 14, wherein the index manager comprises:

an index manager API;

a hash-keyed and sequential file interface;

a B-Tree function; and

a support function.

16. A system for improved access to databases in a computer system, comprising:

a plurality of data sources, the data sources including at least one non-relational data source;

a server coupled to the plurality of data sources, the server capable of providing client access to the plurality of data sources; and

a SPDAL coupled to the server and the at least one non-relational data source, the SPDAL providing relational features to the at least one non-relational data source,

wherein the relational features comprises:

a two-phase commit process;

a process for performing SQL operations from within a single application;

a referential integrity process; and

a process for providing indices for non-relational data source files;

wherein the SPDAL comprises:

a SPDAL ODBC API coupled to the server;

an ODBC Driver Manager coupled to a Generic Access API;

a Generic Driver coupled to the ODBC Driver Manager, the Generic Driver comprising an intercept layer, the intercept layer functioning as a resource manager for the at least one non-relational database during the two-phase commit process;

a SQL Engine coupled to the Generic Driver; and

an index manager coupled to the SQL Engine, the index manager allowing for the creation of indices for the at least one non-relational data source.

17. The system of claim 16, wherein the server is a DB2 DataJoiner, the DB2 DataJoiner comprising a Generic Access API which allows for customized access to data sources not already supported by DataJoiner.

18. The system of claim 16, wherein the index manager comprises:

an index manager API;

a hash-keyed and sequential file interface;

a B-Tree function; and

a support function.

19. A method for performing a two-phase commit process in a computer system, the computer system including a plurality of data sources, the plurality of data sources including at least one non-relational data source, comprises the steps of:

(a) sending a first request to the at least one non-relational data source for commitment to a transaction;

(b) receiving a first reply from the at least one non-relational data source in response to the first request;

(c) sending a second request to another of the plurality of data sources for commitment to the transaction;

(d) receiving a second reply from the other of the plurality of data sources in response to the second request;

(e) determining if the first and second replies both indicate a commitment to the transaction; and

(f) performing the transaction if the first and second replies both indicate a commitment to the transaction.

20. The method of claim 19, further comprising:

(g) indicating a failure of the transaction if the first and second replies do not both indicate a commitment to the transaction.

21. A method for performing SQL operations in a computer system, the computer system including a plurality of data sources, the plurality of data sources including at least one non-relational data source, comprising the steps of:

(a) providing a plurality of metadata tables, the metadata tables describing a content of the at least one non-relational data source; and

(b) utilizing the metadata tables to gain access to the at least one non-relational data source and the other data source wherein the SQL operation is performed from within a single application.

22. A method for performing referential integrity in a computer system, the computer system including a plurality of data sources, the plurality of data sources including at least one non-relational data source, comprising the steps of:

(a) capturing an end user query;

(b) preprocessing the end user query based upon information in metadata tables, the metadata tables describing a content of the at least one non-relational data source;

(c) returning the end user query to the server.

23. The method of claim 22, wherein the preprocessing step (b) comprises:

(b1) determining referential integrity is required for the end user query;

(b2) formulating at least one query to execute and enforce referential constraint for the end user query.

24. The method of claim 22, further comprising the steps of:

(d) processing the end user query on the at least one non-relational data source; and

(e) returning a result of the end user query.

25. The method of claim 24, wherein the processing step (d) comprises:

(d1) translating the at least one query to a command understandable by the at least one non-relational data source; and

(d2) executing the command.

26. A method for performing referential integrity in a computer system, the computer system including a plurality of data sources, the plurality of data sources including at least one non-relational data source, comprising the steps of:

(a) capturing an end user query;

(b) determining referential integrity is required for the end user query;

(c) formulating at least one query to execute and enforce referential constraint for the end user query;

(d) returning the end user query to the server;

(e) determining if the end user query requires data from the at least one non-relational data source;

(f) translating the at least one query to a command understandable by the at least one non-relational data source if data from the at least one non-relational data source is required;

(g) executing the command; and

(h) returning a result of the end user query to the server.

27. A method for providing indices in a computer system, the computer system including a plurality of data sources, the plurality of data sources including at least one non-relational data source, comprising the steps of:

(a) creating indices for the at least one non-relational data source; and

(b) utilizing the indices in executing a query.

28. The method of claim 27, wherein the creating step (a) comprises:

(a1) creating hash-keyed and sequential files for the at least one non-relational data source;

(a2) specifying at least one field of the hash-keyed and/or sequential files; and

(a3) creating an index for the at least one field.

29. The method of claim 27, wherein the utilizing step (b) comprises:

(b1) determining if an index exists which assists the performance of a query;

(b2) retrieving the index if the index exists;

(b3) determining if the index alone satisfies the query; and

(b4) retrieving additional data from the at least one non-relational data source based upon the index if the index alone does not satisfy the query.

30. A method for providing indices in a computer system, the computer system including a plurality of data sources, the plurality of data sources including at least one non-relational data source, comprising the steps of:

(a) creating hash-keyed and sequential files for the at least one non-relational data source;

(b) specifying at least one field of the hash-keyed and/or sequential files;

(c) creating an index for the at least one field;

(d) determining if an index exists which assists the performance of a query;

(e) retrieving the index if the index exists;

(f) determining if the index alone satisfies the query; and

(g) retrieving additional data from the at least one non-relational data source based upon the index if the index alone does not satisfy the query.

31. A computer readable medium with program instructions for performing a two-phase commit process in a computer system, the computer system including a plurality of data sources, the plurality of data sources including at least one non-relational data source, program instructions for:

(a) sending a first request to the at least one non-relational data source for commitment to a transaction;

(b) receiving a first reply from the at least one non-relational data source in response to the first request;

(c) sending a second request to another of the plurality of data sources for commitment to the transaction;

(d) receiving a second reply from the other of the plurality of data sources in response to the second request;

(e) determining if the first and second replies both indicate a commitment to the transaction; and

(f) performing the transaction if the first and second replies both indicate a commitment to the transaction.

32. A computer readable medium with program instructions for performing SQL operations in a computer system, the computer system including a plurality of data sources, the plurality of data sources including at least one non-relational data source, program instructions for:

(a) providing a plurality of metadata tables, the metadata tables describing a content of the at least one non-relational data source; and

(b) utilizing the metadata tables to gain access to the at least one non-relational data source and the other data source wherein the SQL operation is performed from within a single application.

33. A computer readable medium with program instructions for performing referential integrity in a computer system, the computer system including a plurality of data sources, the plurality of data sources including at least one non-relational data source, program instructions for:

(a) capturing an end user query;

(b) preprocessing the end user query based upon information in metadata tables, the metadata tables describing a content of the at least one non-relational data source;

(c) returning the end user query to the server.

34. A computer readable medium with program instructions for performing referential integrity in a computer system, the computer system including a plurality of data sources, the plurality of data sources including at least one non-relational data source, program instructions for:

(a) capturing an end user query;

(b) determining referential integrity is required for the end user query;

(c) formulating at least one query to execute and enforce referential constraint for the end user query;

(d) returning the end user query to the server;

(e) determining if the end user query requires data from the at least one non-relational data source;

(f) translating the at least one query to a command understandably by the at least one non-relational data source;

(g) executing the command; and

(h) returning the end user query to the server.

35. A computer readable medium with program instructions for providing indices in a computer system, the computer system including a plurality of data sources, the plurality of data sources including at least one non-relational data source, program instructions for:

(a) creating indices for the at least one non-relational data source; and

(b) utilizing the indices in executing a query.

RA9-98-071

12/28

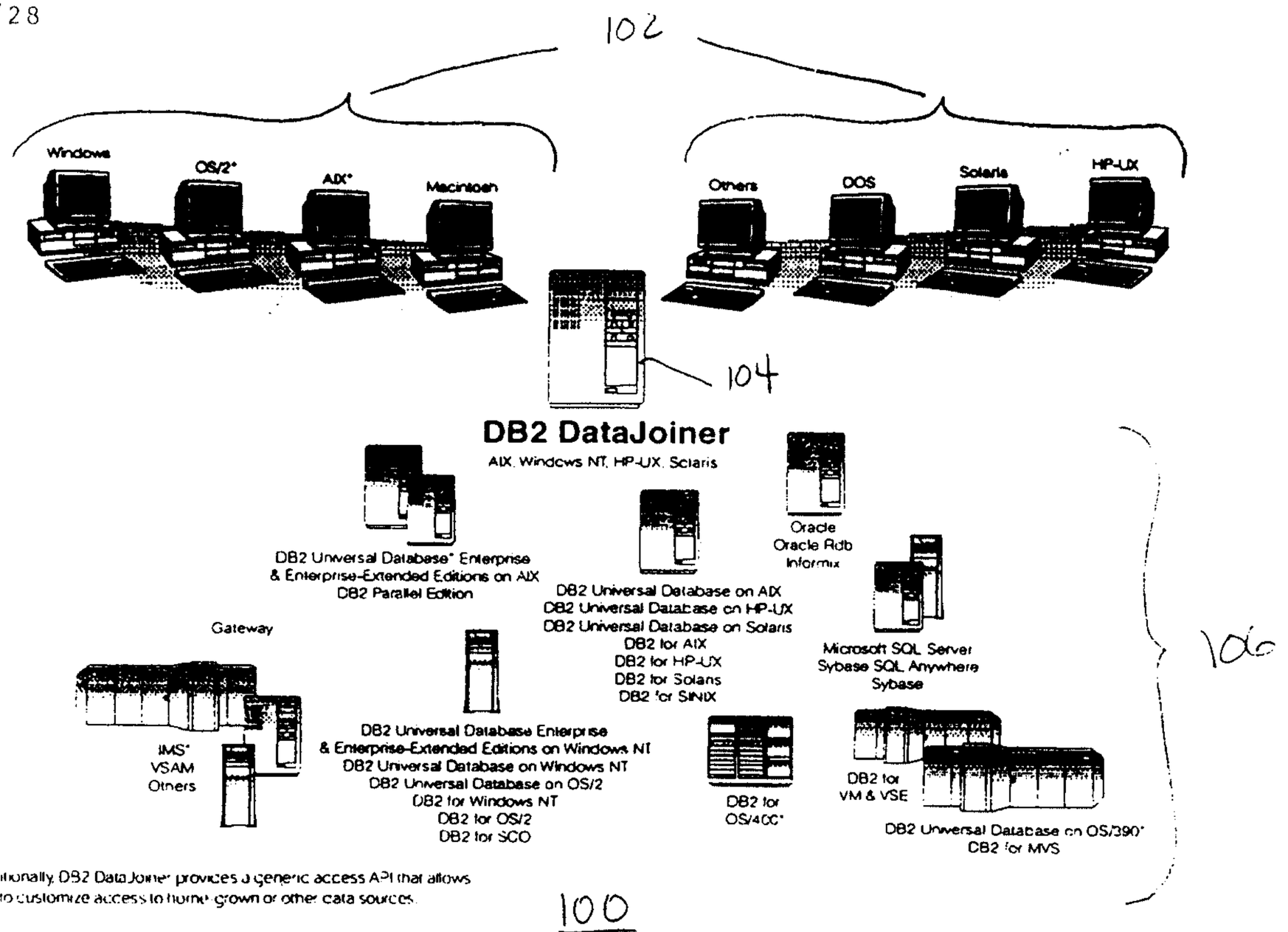


FIGURE 1

PRIOR ART

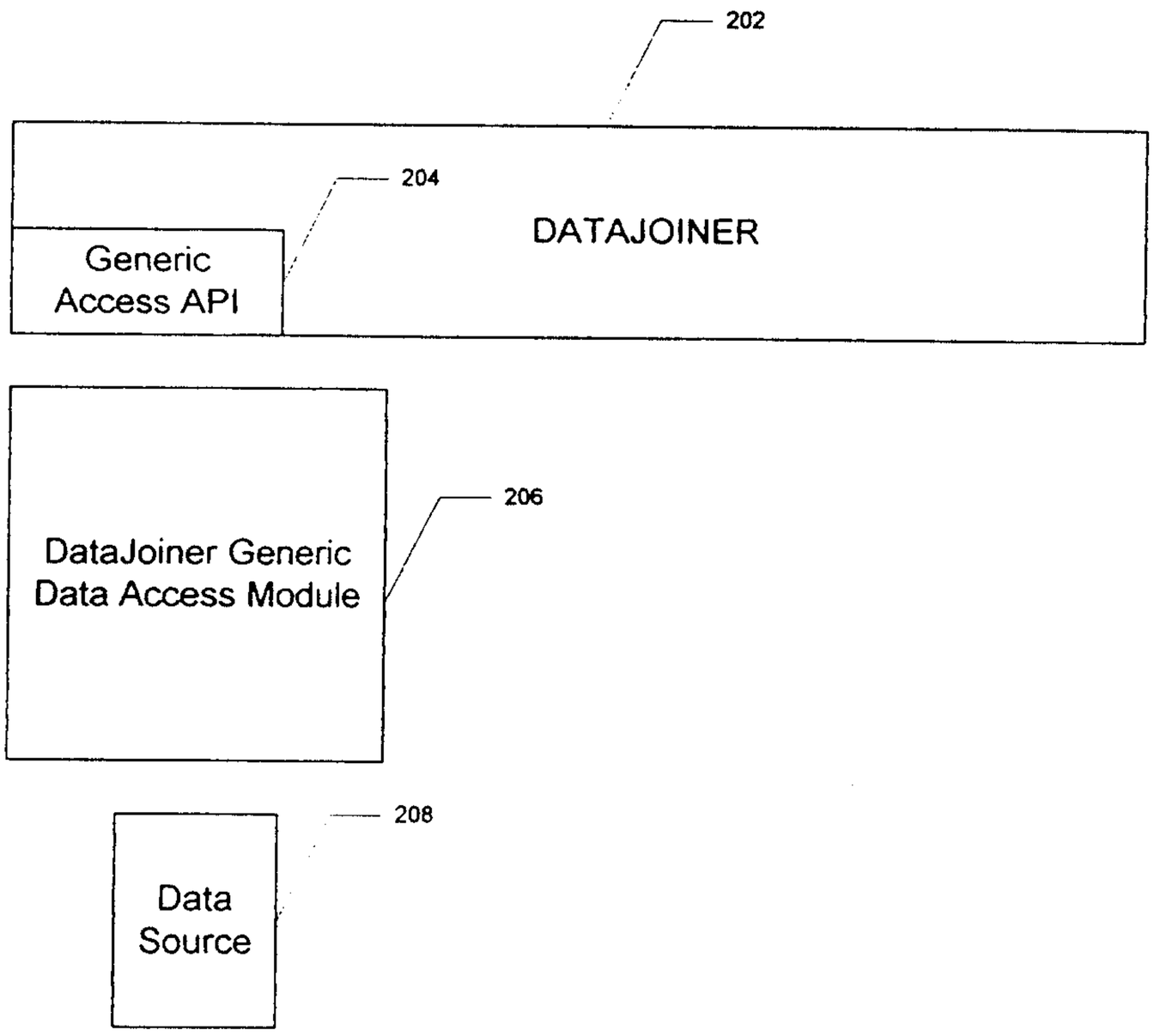


FIGURE 2
PRIOR ART

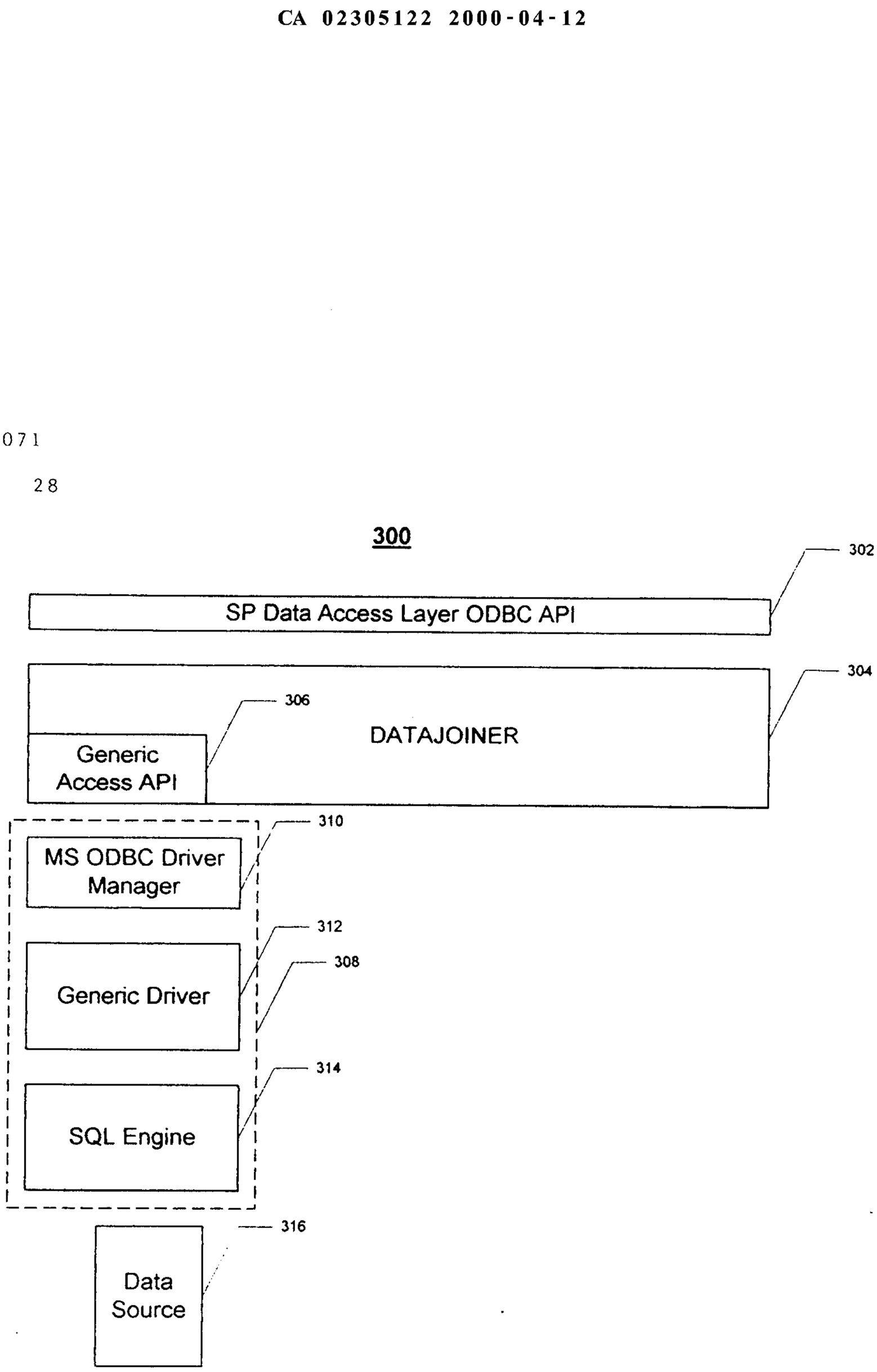

300
302
SP Data Access Layer ODBC API
304
306
DATAJOINER
Generic Access API
310
MS ODBC Driver Manager
312
308
Generic Driver
314
SQL Engine
316
Data Source

FIGURE 3

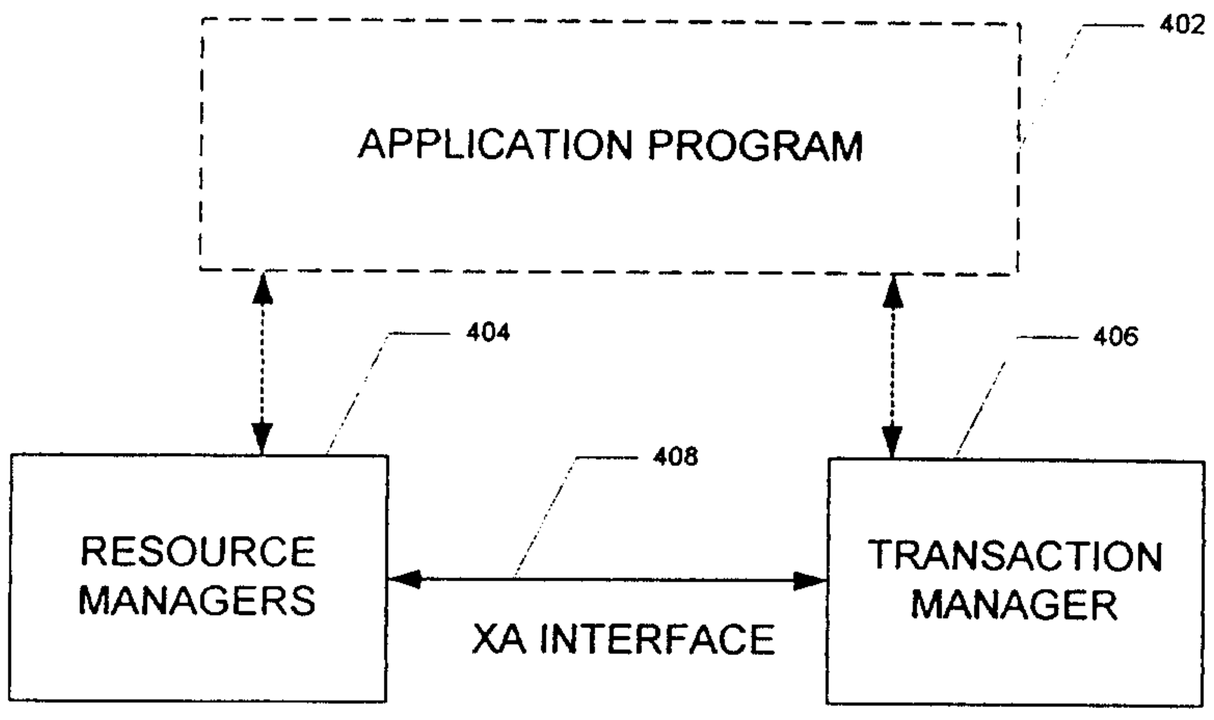

APPLICATION PROGRAM
402
404
406
408
RESOURCE MANAGERS
XA INTERFACE
TRANSACTION MANAGER

FIGURE 4

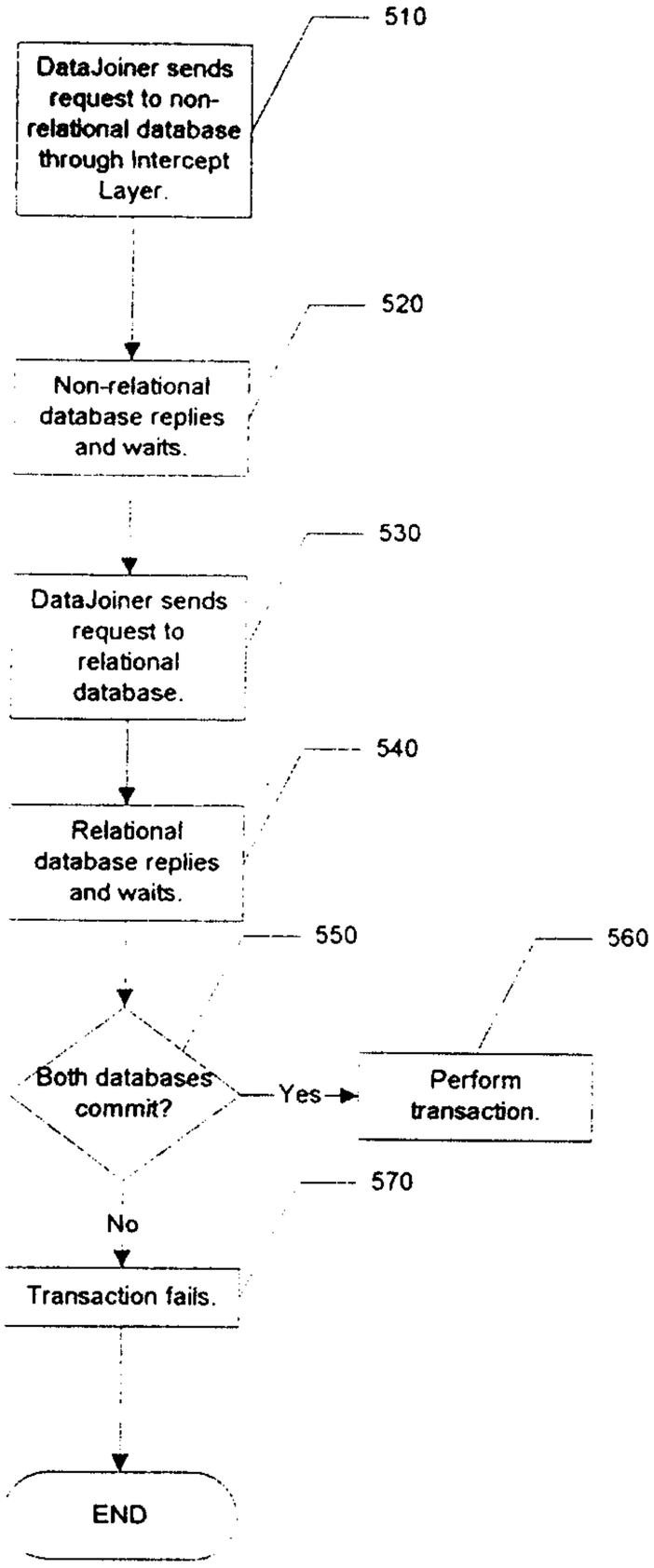

510
DataJoiner sends request to non-relational database through Intercept Layer.
520
Non-relational database replies and waits.
530
DataJoiner sends request to relational database.
540
Relational database replies and waits.
550
Both databases commit?
Yes
560
Perform transaction.
570
No
Transaction fails.
END

FIGURE 5

End User Query Requiring Update to Multiple Data Sources
Using Two-Phase Commit

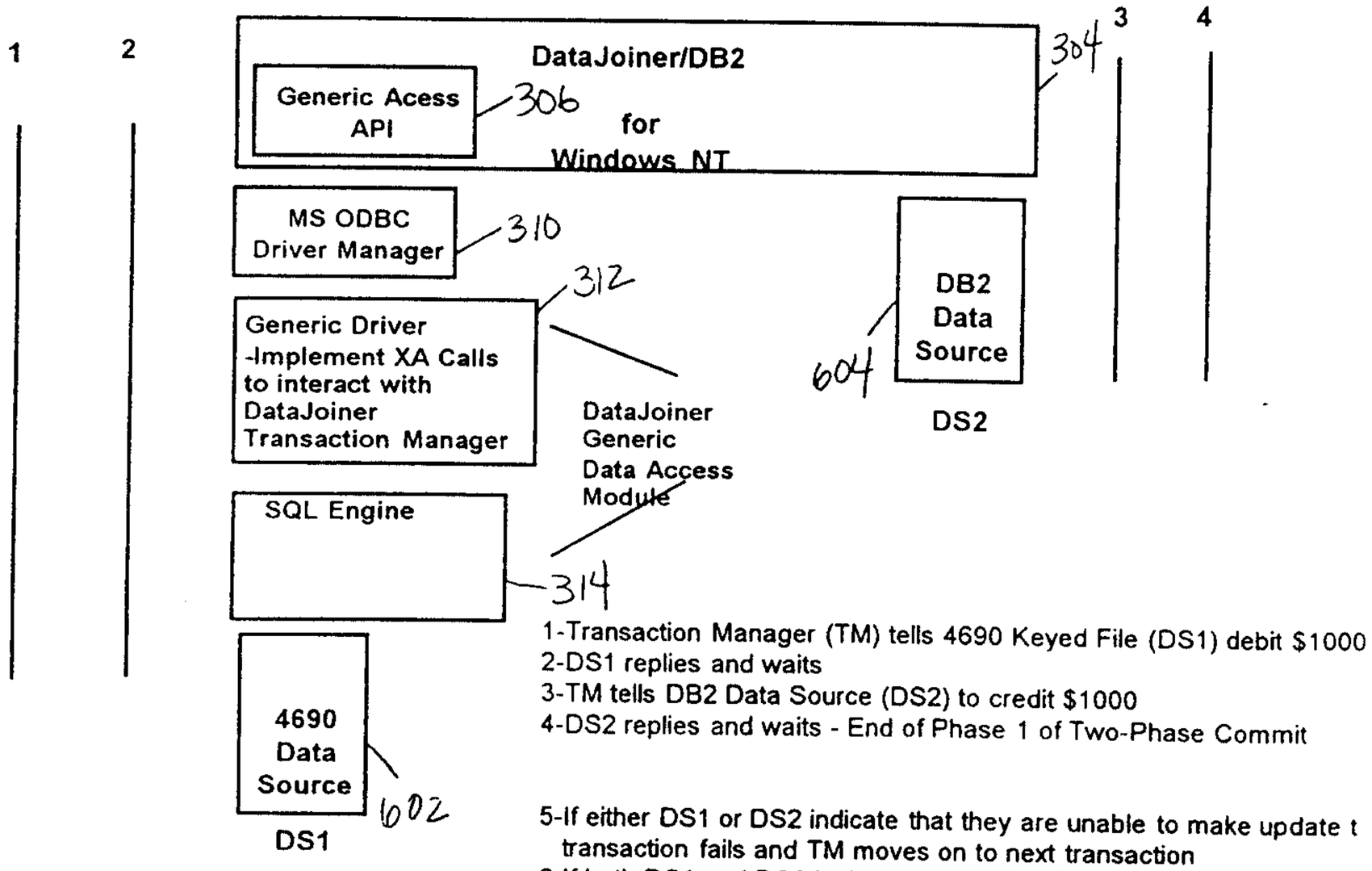


FIG. 6

TABLE 1
Logical Entity Table

Keyword	*Short Description*	*DJ Data Type*	*Acceptable Values*
LogicalEntityName	The name of the logical entity.	varchar(254)	No restrictions.
LogicalEntityID	SPDAL generated numeric unique identifier for this LogicalEntityName. Used for SPDAL operations only.	integer	Used by SPDAL internal operations only, the customer should not modify.
Description	A description of the Logical Entity that is meaningful to the customer. This is not directly used by SPDAL.	varchar(4096)	No restrictions. Not used by SPDAL.

FIGURE 7

TABLE 2
PhysicalEntity Table

Keyword	*Short Description*	*DJ Data Type*	*Acceptable Values*
PhysicalEntityID	SPDAL generated numeric unique identifier for this PhysicalName. Used for SPDAL operations only.	integer	Used by SPDAL internal operations only, the customer should not modify.
LogicalEntityID	SPDAL generated numeric unique identifier for this LogicalEntityName. This column is used to map back to the LogicalEntity table.	integer	Used by SPDAL internal operations only, the customer should not modify.

FIG.7 (Cont')

Table 2 (Con't) FIG.7 (Con't)

	Used for SPDAL operations only		
PhysicalName	Fully qualified name of RDB table or file name without path specification. The PhysicalName could be a 4690 or DDS logical name. For files this includes the file name and extension.	varchar(128)	This is used by the IP only, thus each name must adhere to OS, DDS, 4690, and RDB rules and is dependent on the data store. SPDAL can't verify these restrictions prior to SQL statements.
PhysicalPathName	Used for 4690 or DDS files only. This is the drive letter and path name without the file name and extension.	varchar(254)	This is used by the IP only, thus the value must adhere to OS, 4690, and DDS rules and is dependent on the data store. SPDAL can't verify these restrictions prior to SQL statements. This can be NULL.
DefinedInPhysicalEntityID	This PhysicalEntityID has been defined by another PhysicalEntityID. This is used when multiple physical files have similar physical definitions that are in this table. The underlying file structure, for example as defined in the Attributes table, must be identical. This column is provided as a metadata administration convenience and is not required. This is	integer	Must be an existing PhysicalEntityID. May be NULL.

Table 2 (Con't) FIG.7 (Con't)

	for 4690 and DDS files only.		
DataSource	Type of file system (if an RDB then the RDB vendor) where data is stored. For DDS and 4690 the file organization type is specified.	varchar(50)	Discrete Values: DataJoiner, Oracle, Informix, Sybase, SQLServer, 4690Key, 4690Dir, 4690Seq, DDSKey, DDSBin, DDSSeq.
ServerName	Name of software physical entity providing the data access. This is the server name needed to create the DJ nickname. For RDB this represents a machine on the LAN that will provide the data source. For 4690 and DDS this is a fixed value that directs DataJoiner processing to the correct local ODBC driver.	varchar (18)	Valid DB2 long identifier. For 4690 this is '4690', for DDS this is 'DDS'.
NodeName	This is required for 4690 and DDS only. For 4690 this contains the RCO controller nodename that contains the physical file. For DDS this contains the nodename or rolename that contains the physical file.	varchar(50)	Should be NULL for RDBs. For 4690, this must be ADXLX'nnN' where nn can be 'CC' through 'ZZ'. For DDS this must be a valid DDS nodename or rolename. A nodename can be 1 to 8 characters, rolename is 1 to 8 characters encapsulated within <>, for example <rolename>.
KeySize	Number of bytes of a keyed record that	integer	DDS and 4690 must

Table 2 (Con't) FIG.7 (Con't)

	comprises the key. Valid for 4690 and DDS only.		be > 0 and <= RecordSize.
NumberOfBlocks	Number of blocks to be allocated for a keyed file. Valid for 4690 and DDS only.	integer	Must be greater than RandomizingDivisor and >= 2.
BlockSize	The number of bytes to be allocated per block for a keyed file. Valid for DDS only.	integer	A multiple of 512 to 4096 and must be >= RecordSize + 4.
RandomizingDivisor	Randomizing divisor for the file. This determines how the records are dispersed in the file. Valid for 4690 and DDS only.	integer	DDS must be < NumberOfBlocks, can be 0 (system generated default is used). 4690 must > 0 and < NumberofBlocks.
ChainThreshold	The chaining threshold for the file. This is the number of sectors that are checked to find a keyed record before a warning is sent that the file is becoming inefficient. Valid for 4690 and DDS only.	integer	DDS must < NumberofBlocks. 4690 1 to 255.
HashingAlgorithm	Indicates which hashing algorithm to be used. Valid for 4690 and DDS only.	smallint	0 = PolyNomial, 1 = XOR Rotation, 2 = Simple Folding.
OwnerName	The ID of the owner of the RDB table. Not used with 4690 and DDS. Used with the DataJoiner Create Nickname call.	varchar(128)	Must be NULL for DDS and 4690. For RDB must be a DataJoiner Remote Identifier.
ODBCData	For use by SPDAL only.	varchar(254)	For use by SPDAL only, customer should not modify.
SQLSupport	List of what SQL operations are	varchar(128)	See note below for list of acceptable discrete values.

Table 2 (Con't) FIG.7 (Con't)

	supported for this 4690 or DDS file.		
DJNickname	The DataJoiner Nickname used for this physical file/table name.	char(30)	Max 8 characters with a . followed by a max of 18 characters
AllowCreate	Indicate if SPDAL is allowed to Create this physical file. This is for 4690 and DDS only.	smallint	0 = No, 1 = Yes.
AllowDelete	Indicate if SPDAL is allowed to Delete this physical file. This is for 4690 and DDS only.	smallint	0 = No, 1 = Yes.
DDFrequency	The data distribution frequency for this physical file. This is for 4690 and DDS only.	smallint	0 = distribute on update, 1 = distribute on close, 2 = no data distribution. These are consistent for DDS and 4690.
DDDomain	The data distribution domain (the set of nodes that receive the file updates) for this physical file. This is for 4690 and DDS only.	smallint	0 = mirrored, 1 = broadcast/compound. May be NULL.
BCastDomainName	The name of the broadcast domain. This is valid for DDS only.	char(10)	Must be <= 9 characters. Set to NULL if DDDomain = mirrored or is NULL.
FileLock	The type of file level lock that will be attempted for each SQL statement when SPDAL opens the file. This is for DDS only.	smallint	0 = FDS_FILE_LOCK_EXCLUSIVE, 1 = FDS_FILE_LOCK_SHARED, 2 = FDS_FILE_LOCK_NONE
Remarks	Description of the file	varchar(254)	No restriction, this field is not used by SPDAL directly. The

Table 2 (Con't) FIG.7 (Con't)

			customer can provide meaningful data if they want. This field can be NULL.

FIG.7 (Con't)

Table 3.
RecordDescription Table

Keyword	*Short Description*	*DJ Data Type*	*Acceptable Values*
EntityID	SPDAL generated numeric unique identifier for this physical file name. Used for SPDAL operations only.	integer	Used for SPDAL only, the customer should not modify.
RecordID	SPDAL generated numeric unique identifier for this RecordType. Used for SPDAL operations only.	integer	Used for SPDAL only, the customer should not modify.
RecordName	The name of the record that uniquely identifies the record format.	varchar(254)	Identifier that is meaningful to the customer of what the record is used for.
AttributeID	The ID for the attribute whose value uniquely identifies the record. This is only used for sequential and binary files where multiple record types can be specified.	integer	Used for SPDAL only, the customer should not modify.
AttributeIDFieldValue	The value within the AttributeID that uniquely identifies the record. This is only used for	varchar(32)	Customer generated and must be unique for each RecordName within each PhysicalEntity.

Table 3 (Con't) FIG.7 (Con't)

	sequential and binary files where multiple record types can be specified.		
RecordSize	Number of bytes of the record. For a keyed file there can only be one record type. For sequential and binary files there may be one or more record types.	integer	DDS >= KeySize and <= (BlockSize - 4). 4690 1 to 508. For sequential and binary files on both 4690 and DDS it must be >= 1, and < size of the file. This is essentially a read-only field, this will be calculated by SPDAL based on the values in the Attribute table.

FIG.7 (Con't)

Table 4
Attribute

Keyword	*Short Description*	*DJ Data Type*	EntityID
EntityID	SPDAL generated numeric unique identifier for this physical file name. Used for SPDAL operations only. This is used for 4690 and DDS only.	integer	Used for SPDAL only, the customer should not modify.
RecordID	SPDAL generated numeric unique identifier for this	integer	Used for SPDAL only, the customer should not modify.

Table 4 (Con't) FIG.7 (Con't)

	RecordType. Used for SPDAL operations only. This is used for 4690 and DDS only.		
AttributeID	SPDAL generated numeric unique identifier for this Attribute. Used for SPDAL operations only. This is used for 4690 and DDS only.	integer	Used for SPDAL only, the customer should not modify.
LogicalAttributeNam e	The logical name of this attribute. This name uniquely identifies this attribute.	char(128)	No restriction
PhysicalAttributeNa me	The physical name of this attribute.	varchar(18)	No restriction. This value should be NULL for 4690 and DDS. This is restricted to a DB2 long identifier.
StoredDataType	The data type used to store this attribute in DDS and 4690.	varchar(50)	Please refer to the document *SPDAL Data Type Handling for 4690 and DDS.* This value should be NULL for RDBs.
ReturnedDataType	The data type that is returned to the application for this attribute.	varchar(50)	A valid ODBC datatype, currently SPDAL supports ODBC 2 data types. The exact data types supported depend on DataJoiner ODBC support. The integer values to represent data types will be based off of OA.
PrimaryKey	Indicates if this attribute is involved with the primary key,	smallint	0 = No, 1 = yes

Table 4 (Con't) FIG.7 (Con't)

	which uniquely identifies a record.		
PositionInPrimaryKey	The position, starting with 1 for the first attribute, that this attribute occupies in the primary or foreign key. This is for DDS and 4690 only	smallint	No restriction.
Offset	The starting byte position within the record.	integer	No restriction.
BitOffset	The bit offset into a field. This is just for support of bit manipulation, for example, INDICAT fields on 4690.	smallint	No restriction.
MinimumValue	The smallest acceptable value. Applicable for numeric fields only.	Decimal(31,5)	No restriction. This can be a negative number. For 4690 applications , 2 digits after the decimal point are supported.
MaximumValue	The largest acceptable value. Applicable for numeric fields only.	Decimal(31,5)	No restriction. This can be a negative number. For 4690 applications, 2 digits after the decimal point are supported.
DefaultValue	Value used if attribute omitted on API call. For example, an INSERT without values specified for each column/field. This is only used for 4690 and DDS.	varchar(254)	No restriction. The value will always be provided in character form and SPDAL will convert it to the proper StoredDataType.
NullsAllowed	Indicates if NULL values can be used for this attribute. This	smallint	0 = No, 1 = Yes

Table 4 (Con't) FIG.7 (Con't)

	is only used for 4690 and DDS.		
ValueOfNULL	Indicates the value that represents a NULL. This is only used for 4690 and DDS.	varchar(30)	No restriction. The value will be specified as character and SPDAL will convert it to the proper StoredDataType.
ValueOfTRUE	Indicates the value that represents TRUE. This is only used for 4690 and DDS.	varchar(30)	No restriction. The value will be specified as character and SPDAL will convert it to the proper StoredDataType.
ValueOfFALSE	Indicates the value that represents a FALSE. This is only used for 4690 and DDS.	varchar(30)	No restriction. The value will be specified as character and SPDAL will convert it to the proper StoredDataType.
UniqueValueRequire d	Indicates if the values of this attribute within the physical entity must all be unique. This is only used for 4690 and DDS.	smallint	0 = No, 1 = Yes
CharLength	Maximum length for character data types. This is only used for 4690 and DDS.	integer	No restriction. For all data types other than char and varchar it is set to NULL.
Precision	Number of digits of mantissa precision. This is only for 4690 and DDS.	integer	For CHAR or VARCHAR, must be set to same value as CharLength. For NUMERIC it is the number of total digits. For any other data type it is set to NULL since it is hardcoded.

Table 4 (Con't) FIG.7 (Con't)

Radix	Reserved for future use. This is only for 4690 and DDS.	integer	NULL.
Scale	Total number of significant digits to the right of the decimal point. This is only for 4690 and DDS.	integer	No restriction for NUMERIC, NULL for all other data types.
Scope	Reserved for future use. This is only for 4690 and DDS.	smallint	NULL.
AccessData	Used by SPDAL only, not set by the customer. This is only for 4690 and DDS.	varchar(254)	Not available for customer use.
ParentEntityID	Used to identify the parent attribute of this subattribute. This column should only be set if BitOffset is set, meaning that this row describes bit data within a parent attribute. This is a recursive table lookup.	integer	Used for SPDAL only, the customer should not modify.
ParentRecordID	Used to identify the parent attribute of this subattribute. This column should only be set if BitOffset is set, meaning that this row describes bit data within a parent attribute. This is a recursive table lookup.	integer	Used for SPDAL only, the customer should not modify.
ParentAttributeID	Used to identify the parent attribute of this subattribute.	integer	Used for SPDAL only, the customer should not modify.

Table 4 (Con't) FIG.7 (Con't)

	This column should only be set if BitOffset is set, meaning that this row describes bit data within a parent attribute. This is a recursive table lookup.		
Remarks	Any user description of the use of the column. SPDAL does not use this field.	varchar(254)	No restriction.

FIG.7 (Cont)

Table 5

Valid Attribute Values Table

Column Name	*Short Description*	*DJ Data Type*	*Acceptable Values*
EntityID	SPDAL generated numeric unique identifier for this physical file name . Used for SPDAL operations only. Used for DDS and 4690 only.	integer	Used for SPDAL only, the customer should not modify.
RecordID	SPDAL generated numeric unique identifier for this RecordType. Used for SPDAL operations only. Used for DDS and 4690 only.	integer	Used for SPDAL only, the customer should not modify.
AttributeID	SPDAL generated numeric unique identifier for this Attribute. Used for SPDAL operations only. Used for DDS and 4690 only.	integer	Used for SPDAL only, the customer should not modify.
ValidValue	One of the discrete valid values that may be specified by the calling application for this attribute.	varchar(254)	No restriction. The value is specified as character, SPDAL will convert it to the proper StoredDataType.

FIG.7 (Con't)

Table 6.

ForeignKey Table

Column Name	*Short Description*	*DJ Data Type*	*Acceptable Values*
ForeignKeyName	Gives a name to the relationship described.	varchar(18)	Must adhere to DB2 long identifier rules.
ParentEntityID	PhysicalEntityID name of the table or file that contains the parent side of the relationship.	integer	Used for SPDAL only, the customer should not modify.
ChildEntityID	PhysicalEntityID table or file that contains the child side of the relationship.	integer	Used for SPDAL only, the customer should not modify.
ForeignKeyAttributes	List of the AttributeIDs that comprise the foreign key in the order they are defined in the primary key of the parent record.	varchar(128)	Must be a list of AttributeIDs that correspond to the same data types and order as the Primary key in the parent table.
DeleteRule	Defines the handling of foreign keys (child) upon the delete of the primary key (parent)	integer	NULL for SPDAL R1. For SPDAL R2 0 = Restrict, 1 = Cascade, 2 = set NULL.
UpdateRule	Defines the handling of foreign keys (child) upon the update of the primary key (parent)	integer	NULL for SPDAL R1.

Table 7
Index Table

FIG.7 (Con't)

Column Name	*Short Description*	*DJ Data Type*	*Acceptable Values*
IndexID	SPDAL generated numeric unique identifier for this index name. Used for SPDAL operations only. Used for 4690 and DDS only.	integer	No restriction. This is set by SPDAL and should not be altered by the customer.
PhysicalEntityID	SPDAL generated numeric unique identifier for this physical file name. Used for SPDAL operations only. Used for 4690 and DDS only.	integer	Used for SPDAL only, the customer should not modify.
RecordID	SPDAL generated numeric unique identifier for this RecordType. Used for SPDAL operations only.	integer	Used for SPDAL only, the customer should not modify.
AttributeID	SPDAL generated numeric unique identifier for this Attribute. Used for SPDAL operations only.	integer	Used for SPDAL only, the customer should not modify.
SequenceInIndex	Order in which this attribute appears in the index.	smallint	For SPDAL R1, this is fixed at 1, the customer should not modify.
IndexName	A descriptive name of the index.	varchar(128)	No restriction.
IndexFileName	Fully qualified name (drive, path, file name and extension) of the file that contains the index values.	varchar(255)	This is used by the Index Manager, thus the value must adhere to OS file naming rules and is dependent on the data store. SPDAL can't verify these restrictions prior to creating the index file.
UniqueIndicator	Indicates if unique values are required for the index field..	smallint	0 = No, 1 = yes

RA9-98-071
25/28

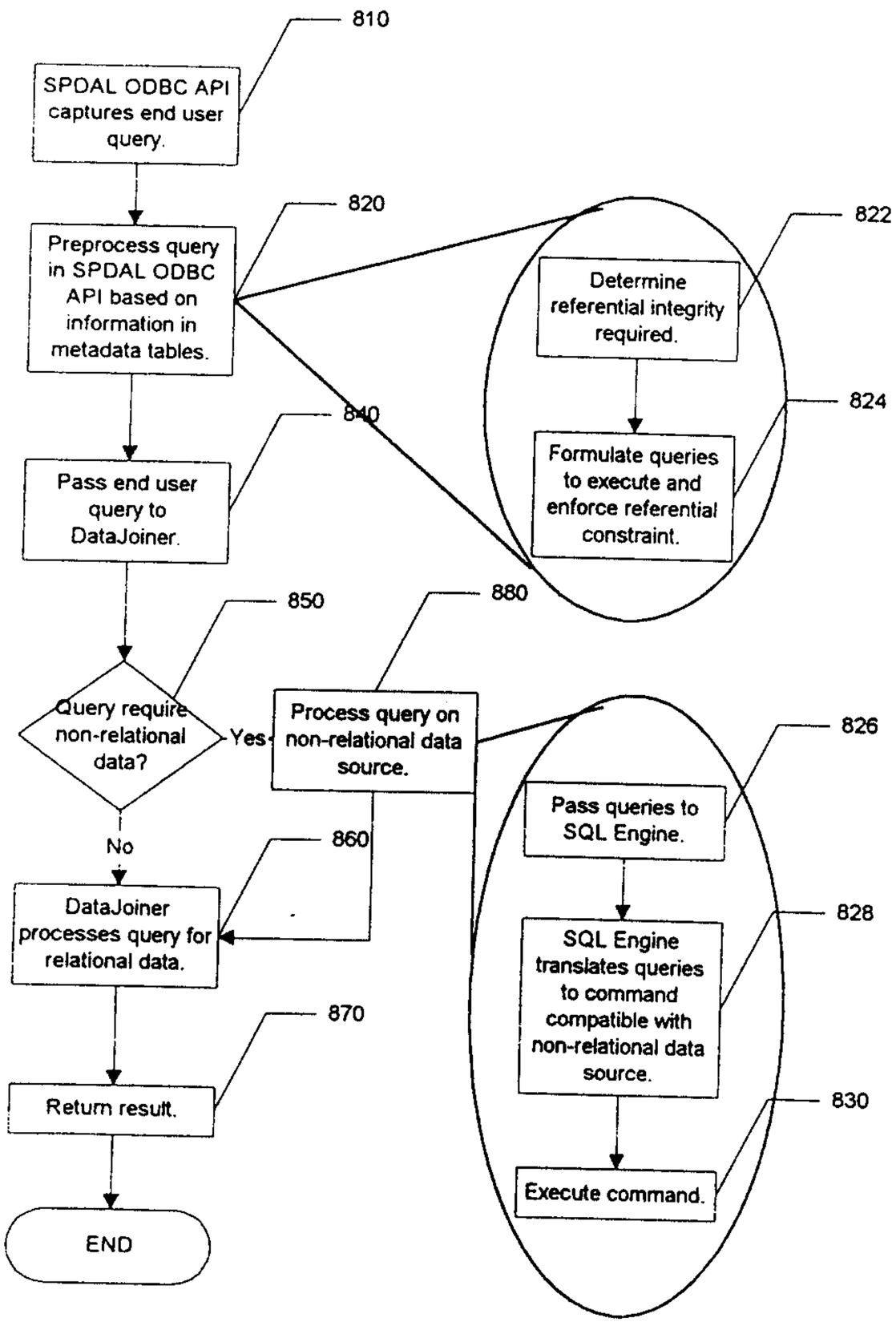


FIGURE 8

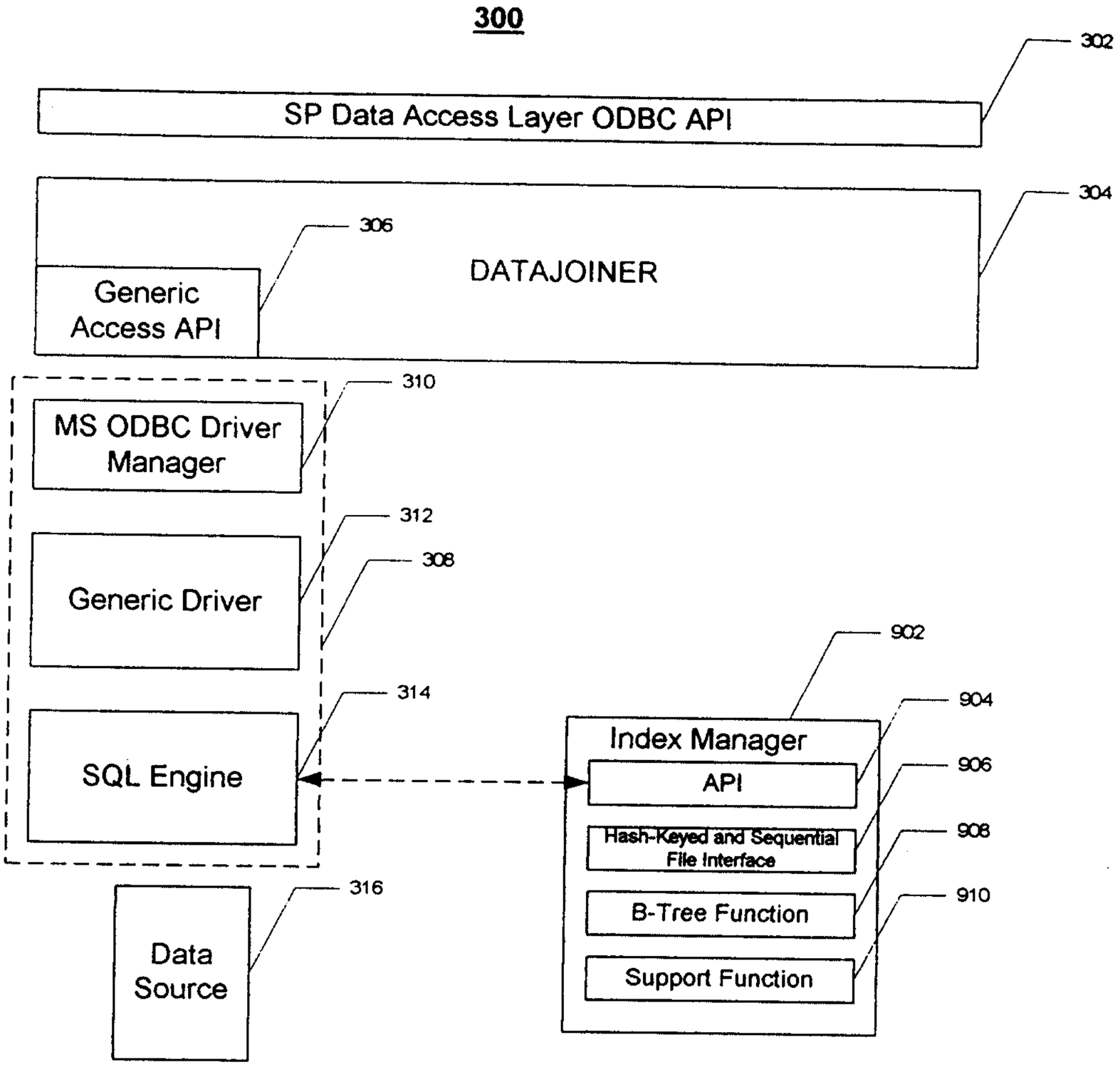

300
302
SP Data Access Layer ODBC API
304
306
DATAJOINER
Generic Access API
310
MS ODBC Driver Manager
312
Generic Driver
308
314
SQL Engine
316
Data Source
902
Index Manager
904
API
906
Hash-Keyed and Sequential File Interface
908
B-Tree Function
910
Support Function

FIGURE 9

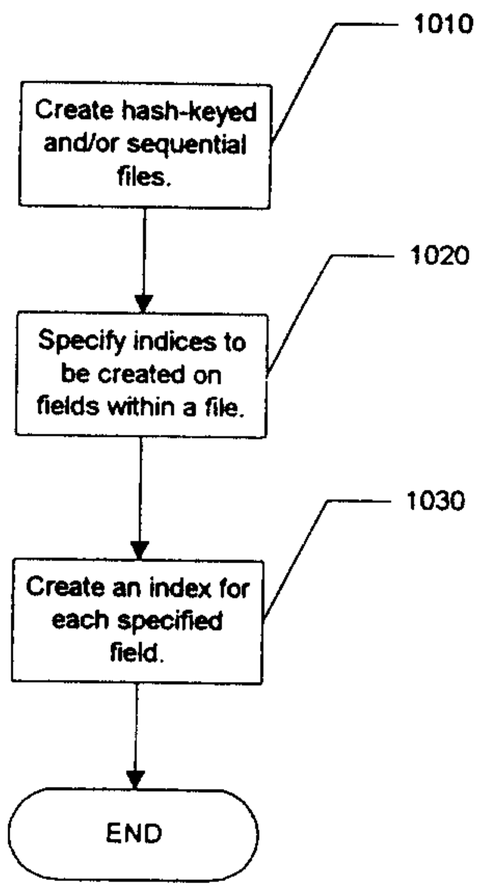

1010
Create hash-keyed and/or sequential files.
1020
Specify indices to be created on fields within a file.
1030
Create an index for each specified field.
END

FIGURE 10

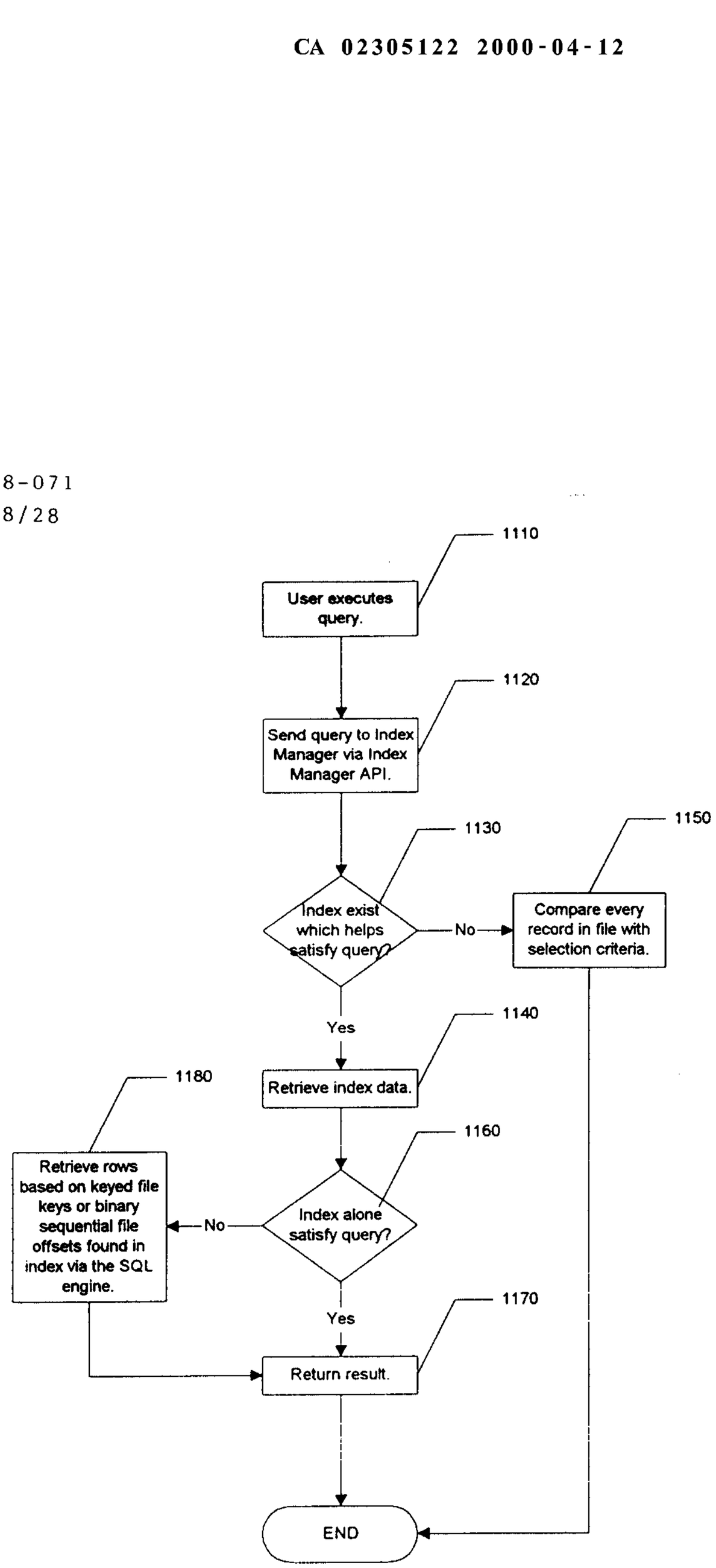
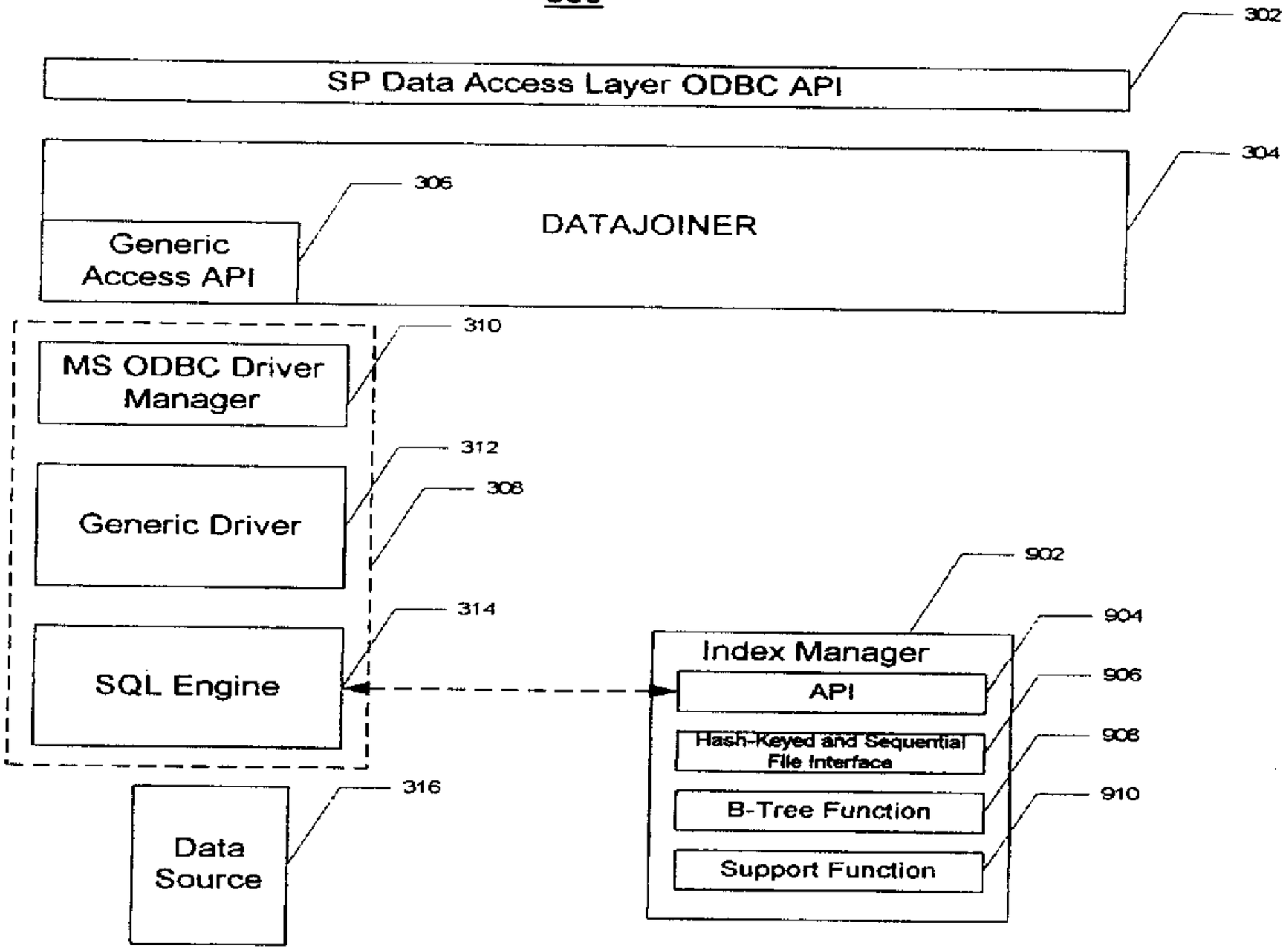

1110
User executes query.
1120
Send query to Index Manager via Index Manager API.
1130
Index exist which helps satisfy query?
No
1150
Compare every record in file with selection criteria.
Yes
1140
Retrieve index data.
1160
Index alone satisfy query?
No
1180
Retrieve rows based on keyed file keys or binary sequential file offsets found in index via the SQL engine.
Yes
1170
Return result.
END

FIGURE 11


300
302
SP Data Access Layer ODBC API
304
306
Generic
Access API
DATAJOINER
310
MS ODBC Driver
Manager
312
Generic Driver
308
314
SQL Engine
316
Data
Source
902
Index Manager
904
API
906
Hash-Keyed and Sequential
File Interface
908
B-Tree Function
910
Support Function