

Document made available under the Patent Cooperation Treaty (PCT)

International application number: PCT/US2006/014196

International filing date: 14 April 2006 (14.04.2006)

Document type: Certified copy of priority document

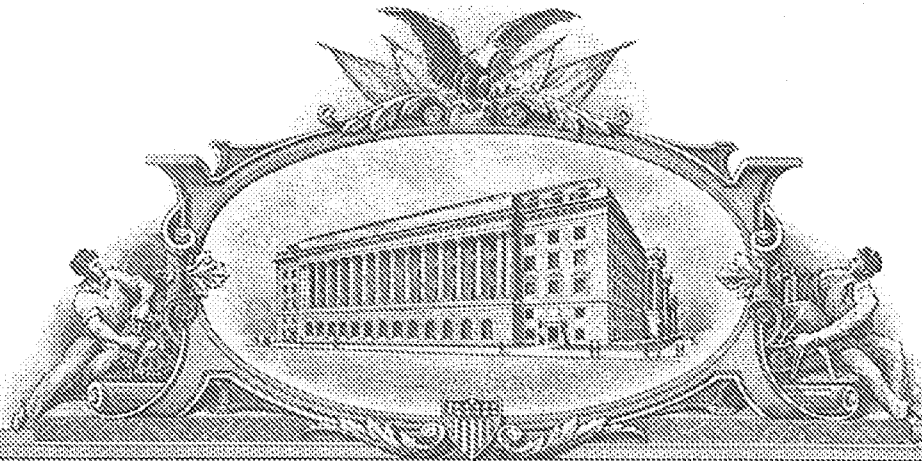
Document details: Country/Office: US
Number: 11/108,204
Filing date: 18 April 2005 (18.04.2005)

Date of receipt at the International Bureau: 15 June 2006 (15.06.2006)

Remark: Priority document submitted or transmitted to the International Bureau in compliance with Rule 17.1(a) or (b)



World Intellectual Property Organization (WIPO) - Geneva, Switzerland
Organisation Mondiale de la Propriété Intellectuelle (OMPI) - Genève, Suisse



THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE

United States Patent and Trademark Office

June 05, 2006

THIS IS TO CERTIFY THAT ANNEXED HERETO IS A TRUE COPY FROM THE RECORDS OF THE UNITED STATES PATENT AND TRADEMARK OFFICE OF THOSE PAPERS OF THE BELOW IDENTIFIED PATENT APPLICATION THAT MET THE REQUIREMENTS TO BE GRANTED A FILING DATE.

APPLICATION NUMBER: *11/108,204*

FILING DATE: *April 18, 2005*

RELATED PCT APPLICATION NUMBER: *PCT/US06/14196*

THE COUNTRY CODE AND NUMBER OF YOUR PRIORITY APPLICATION, TO BE USED FOR FILING ABROAD UNDER THE PARIS CONVENTION, IS *US11/108,204*



Certified by

Under Secretary of Commerce
for Intellectual Property
and Director of the United States
Patent and Trademark Office

17712
041805

17712

US PTO

PTO/SB/05 (09-04)

Approved for use through 07/31/2006. OMB 0651-0032
U.S. Patent and Trademark Office, U.S. DEPARTMENT OF COMMERCE

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

UTILITY
PATENT APPLICATION
TRANSMITTAL

(Only for new nonprovisional applications under 37 CFR 1.53(b))

Attorney Docket No.	oracle01.036
First Inventor	Chong, et al.
Title	Integrating RDF data into a relational database system
Express Mail Label No.	EV 516638307 US

APPLICATION ELEMENTS

See MPEP chapter 600 concerning utility patent application contents.

1. ☒ **Fee Transmittal Form** (e.g., PTO/SB/17)
(Submit an original and a duplicate for fee processing)
2. ☐ **Applicant claims small entity status.**
See 37 CFR 1.27.
3. ☒ **Specification** [Total Pages 45]
Both the claims and abstract must start on a new page
(For information on the preferred arrangement, see MPEP 608.01(a))
4. ☒ **Drawing(s)** (35 U.S.C. 113) [Total Sheets 20]
5. **Oath or Declaration** [Total Sheets 5]
 - a. ☒ Newly executed (original or copy)
 - b. ☐ A copy from a prior application (37 CFR 1.63(d))
(for continuation/divisional with Box 18 completed)
 - i. ☐ **DELETION OF INVENTOR(S)**
Signed statement attached deleting inventor(s)
name in the prior application, see 37 CFR 1.63(d)(2) and 1.33(b).
6. ☐ **Application Data Sheet.** See 37 CFR 1.76
7. ☐ **CD-ROM or CD-R** in duplicate, large table or
Computer Program (Appendix)
☐ Landscape Table on CD
8. ☒ **Nucleotide and/or Amino Acid Sequence Submission**
(if applicable, items a. - c. are required)
 - a. ☐ Computer Readable Form (CRF)
 - b. ☐ Specification Sequence Listing on:
 - i. ☐ CD-ROM or CD-R (2 copies); or
 - ii. ☐ Paper
 - c. ☐ Statements verifying identity of above copies

ADDRESS TO:

Commissioner for Patents
P.O. Box 1450
Alexandria VA 22313-1450

ACCOMPANYING APPLICATION PARTS

9. ☒ **Assignment Papers** (cover sheet & document(s))
Name of Assignee Oracle International
Corporation
10. ☐ **37 CFR 3.73(b) Statement** (when there is an assignee) ☒ **Power of Attorney**
11. ☐ **English Translation Document** (if applicable)
12. ☐ **Information Disclosure Statement** (PTO/SB/08 or PTO-1449)
☐ Copies of citations attached
13. ☐ **Preliminary Amendment**
14. ☒ **Return Receipt Postcard** (MPEP 503)
(Should be specifically itemized)
15. ☐ **Certified Copy of Priority Document(s)**
(if foreign priority is claimed)
16. ☐ **Nonpublication Request** under 35 U.S.C. 122(b)(2)(B)(i).
Applicant must attach form PTO/SB/35 or equivalent.
17. ☐ Other: _____

18. If a CONTINUING APPLICATION, check appropriate box, and supply the requisite information below and in the first sentence of the specification following the title, or in an Application Data Sheet under 37 CFR 1.76:

☐ Continuation ☐ Divisional ☐ Continuation-in-part (CIP) of prior application No.: _____

Prior application information: Examiner _____ Art Unit: _____

19. CORRESPONDENCE ADDRESS

☒ The address associated with Customer Number: 25247 OR ☒ Correspondence address below

Name	Gordon E. Nelson, Patent Attorney, PC				
Address	57 Central Street, P.O. Box 782				
City	Rowley	State	MA	Zip Code	01969
Country	US	Telephone	978-948-7632	Fax	866-723-3059

Signature	<u>Gordon E. Nelson</u>	Date	April 18, 2005
Name (Print/Type)	Gordon E. Nelson	Registration No. (Attorney/Agent)	30,093

This collection of information is required by 37 CFR 1.53(b). The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.11 and 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

113277 U.S. PTO
11/108204

EV516638307US

17712
041805

PTO/SB/17 (12-04v2)

Approved for use through 07/31/2006. OMB 0651-0032

U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

Effective on 12/08/2004.

Fees pursuant to the Consolidated Appropriations Act, 2005 (H.R. 4818).

FEE TRANSMITTAL

For FY 2005

☐ Applicant claims small entity status. See 37 CFR 1.27

TOTAL AMOUNT OF PAYMENT (\$) 3000

Complete if Known

Application Number	
Filing Date	
First Named Inventor	CHONG, et al.
Examiner Name	
Art Unit	
Attorney Docket No.	oracle01.036

METHOD OF PAYMENT (check all that apply)

- ☒ Check ☐ Credit Card ☐ Money Order ☐ None ☐ Other (please identify): _____
- ☒ Deposit Account Deposit Account Number: 501315 Deposit Account Name: Gordon E. Nelson, Patent Atty
- For the above-identified deposit account, the Director is hereby authorized to: (check all that apply)
- ☐ Charge fee(s) indicated below ☐ Charge fee(s) indicated below, except for the filing fee
- ☒ Charge any additional fee(s) or underpayments of fee(s) under 37 CFR 1.16 and 1.17 ☒ Credit any overpayments

WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.

FEE CALCULATION

1. BASIC FILING, SEARCH, AND EXAMINATION FEES

Application Type	FILING FEES		SEARCH FEES		EXAMINATION FEES		Fees Paid (\$)
	Fee (\$)	Small Entity Fee (\$)	Fee (\$)	Small Entity Fee (\$)	Fee (\$)	Small Entity Fee (\$)	
Utility	300	150	500	250	200	100	1000
Design	200	100	100	50	130	65	
Plant	200	100	300	150	160	80	
Reissue	300	150	500	250	600	300	
Provisional	200	100	0	0	0	0	

2. EXCESS CLAIM FEES

Fee Description	Fee (\$)	Small Entity Fee (\$)
Each claim over 20 (including Reissues)	50	25
Each independent claim over 3 (including Reissues)	200	100
Multiple dependent claims	360	180

Total Claims 44 - 20 or HP = 24 x 50 = 1200

HP = highest number of total claims paid for, if greater than 20.

Indep. Claims 7 - 3 or HP = 4 x 200 = 800

HP = highest number of independent claims paid for, if greater than 3.

3. APPLICATION SIZE FEE

If the specification and drawings exceed 100 sheets of paper (excluding electronically filed sequence or computer listings under 37 CFR 1.52(e)), the application size fee due is \$250 (\$125 for small entity) for each additional 50 sheets or fraction thereof. See 35 U.S.C. 41(a)(1)(G) and 37 CFR 1.16(s).

Total Sheets Extra Sheets Number of each additional 50 or fraction thereof Fee (\$) Fee Paid (\$)

 - 100 = / 50 = (round up to a whole number) x =

4. OTHER FEE(S)

Non-English Specification, \$130 fee (no small entity discount)

Other (e.g., late filing surcharge):

Fees Paid (\$)

SUBMITTED BY

Signature	<u>Gordon E. Nelson</u>	Registration No. (Attorney/Agent)	30,093	Telephone	978-948-7632
Name (Print/Type)	Gordon E. Nelson	Date	April 18, 2005		

This collection of information is required by 37 CFR 1.136. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 30 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

Patent Application for

5 *Integrating RDF data into a relational database system*

Inventors:

Eugene Inseok Chong

Souripriya Das

George Eadon

Jagannathan Srinivasan

15

Assignee: Oracle International Corporation

Integrating RDF data into a relational database system

Cross references to related patent applications

5 The subject matter of this patent application is closely related to the subject matter of patent application USSN xx/xxx,xxx, *Rewriting table functions as SQL strings*, which has the same inventors and assignee as the present patent application and is being filed on even date with this application. USSN xx/xxx,xxx is further incorporated by reference into this patent application for all purposes.

10

Background of the invention

1. Field of the invention

The invention concerns the representation of semantic knowledge by the Resource Description Framework, or RDF, and more specifically concerns the integration of data represented by RDF into a relational database system.

15

2. Description of related art: FIGs. 1-3

RDF is a language that was originally developed for representing information (metadata) about resources in the World Wide Web. It may, however, be used for representing information about absolutely anything. When information has been specified using the generic RDF format, it may be automatically consumed by a diverse set of applications.

20

FIGs. 1-3 provide an overview of RDF. *Facts* in RDF are represented by RDF *triples*. Each RDF triple represents a fact and is made up of three parts, a *subject*, a *predicate*, (sometimes termed a *property*), and an *object*. For example, the fact represented by the English sentence “John is 24 years old” is represented in RDF by the subject, predicate, object triple <‘John’, ‘age’, ‘24’>, with ‘John’ being the subject, ‘age’ being the predicate, and ‘24’ being the object. In current RDF, the values of subjects and predicates must ultimately resolve to universal resource identifiers (URIs). The values of objects may be literal values such as numbers or character strings. The interpretations given to the members of the triple are determined by the application that is consuming it.

25

30

RDF triples may be represented as a graph as shown at 109 in FIG. 1. The subject is represented by a node 103, the object by another node 107, and the predicate by arrow 104 connecting the subject node to the object node. A subject may of course be related to more than one object, as shown with regard to "Person" 103. Each entity in an RDF triple is represented by a World Wide Web Uniform Resource Identifier (URI) or a literal value. For example, the subject "John" is identified by the URI for his contact information. In RDF triple 117, the value of John's age is the literal value 24. In the following general discussion of RDF, the URIs will be replaced by the names of the entities they represent. For a complete description of RDF, see Frank Manola and Eric Miller, *RDF Primer*, published by W3C and available in September, 2004 at www.w3.org/TR/rdf-primer/. The *RDF Primer* is hereby incorporated by reference into the present patent application.

An RDF representation of a set of facts is termed in the following an *RDF model*. A simple RDF model *Reviewers* is shown at 101 in FIG. 1. The model has two parts: RDF data 113 and RDF schema 111. RDF schema 111 is made up of RDF triples that provide the definitions needed to interpret the triples of RDF data 113. Schema triples define classes of entities and predicates which relate classes of entities. A property definition for the predicate *age* is shown at 112. As shown there, a predicate definition consists of two RDF triples for which the predicate is the subject. One of the triples, which has the built-in domain predicate, indicates what kind of entities must be subjects for the predicate. Here, it is entities belonging to the class *person*. The other triple indicates what kinds of entities must be objects of the predicate; here, it is values of an integer type called *xsd:int*. Schema 111 uses the *SubclassOf* predicate 110 to define a number of subclasses of entities belonging to the class *person*. Also defined are *conference* and *university* classes of entities, together with predicates that relate these entities to each other. Thus, an entity of class *person* may be a chairperson of a conference and an entity of class reviewer may be a reviewer for a conference. Also belonging to Schema 111 but not shown there is the built-in RDF predicate *rdf:type*. This predicate defines the subject of a triple that includes the *rdf:type* predicate as an instance of the class indicated by the object. As will be explained in more detail, *RDF rules* determine logical relationships between classes. For example, a built-in RDF rule states that the *SubclassOf* relationship is transitive: if A is a subclass of B and B a subclass of C, then A is a subclass of C. Thus, the class *faculty* is a subclass of *person*.

The data triples to which schema 111 applies are shown at 113; they have the general pattern <individual entity>, <predicate>, <object characterizing the individual entity>. Thus, triple 115 indicates that ICDE 2005 is an entity characterized as belonging to the class CONFERENCE and triple 117 indicates that JOHN is characterized by having the age 24. Thus, RDF data 113 contains the following triples about John:

John has an Age of 24 ;

John belongs to the subclass Ph.D.Student;

John is a ReviewerOf ICDE 2005.

None of these triples states that John is a Person; however, the fact that he is a Person and a Reviewer is inferred from the fact that he is stated to be a Ph.D.Student, which is defined in schema 111 as a subclass of both Person and Reviewer. Because the Subclassof predicate is transitive, the fact that John is a Ph.D Student means that he is a potential subject of the Age and ReviewerOf properties.

For purposes of the present discussion RDF models are best represented as lists of RDF triples instead of graphs. FIG. 2 shows a table of triples 201 which lists triples making up schema 111 and a table of triples 203 which lists triples making up RDF data 113. At the bottom of FIG. 2 is an *RDF Pattern* 205. An RDF pattern is a construct which is used to query RDF triples. There are many different ways of expressing RDF patterns; what follows is a typical example. When RDF pattern 205 is applied to RDF model 101, it will return a subgraph of RDF model 101 which includes all of the reviewers of conference papers who are Ph.D students. The pattern is made up of one or more patterns 207 for RDF triples followed by an optional filter which further restricts the RDF triples identified by the pattern. The identifiers beginning with ? are variables that represent values in the triples belonging to the subgraph specified by the RDF pattern. Thus, the first pattern 207(1) specifies every Reviewer for every Conference indicated in the RDF data 203; the second pattern 207(2) specifies every Reviewer who belongs to the subclass Ph.D.Student, and the third pattern 207(3) specifies every Person for which an Age is specified. The result of the application of these three patterns to RDF data 203 is the intersection of the sets of persons specified by each of the patterns, that is, the intersection of the set of reviewers and the set of Ph.D.

Students of any age. The intersection is John, Tom, Gary, and Bob, who are indicated by the triples in data 203 as being both Ph.D students and reviewers.

The manner in which entities in an RDF model relate to each other can be modified by applying
5 RDF *rules*. An example RDF rule is shown at 301 in FIG. 3. Rule 301 is contained in a rulebase
which, as shown at 303, has the name `rb`. The rule has a name, `chairpersonRule`, which is
shown at 305. As will be explained in detail later, the rule specifies how the class of `Persons`
who are conference chairpersons relates to the class of `Reviewers` for the conference. Rule body
310 has a left-hand side 307 specifying the rule's *antecedent* and a right-hand side 311 specifying
10 the rule's *consequent*. The rule states that if an entity satisfies the conditions established for the left-
hand side 307 (the antecedent), it also satisfies the conditions established for the right-hand side 311
(the consequent). The antecedent and the consequent are specified by RDF patterns. The RDF
pattern for left-hand side 307 specifies any `Person` (`?r`) in the model who is a chairperson of any
`Conference` (`?c`) in the model; the RDF pattern for right-hand side 311 specifies that any
15 such person is also a reviewer for that conference.

RDF pattern 312 shows the effect of rule 301. The pattern's triple specifies RDF triples which have
the `ReviewerOf` predicate. Without rule 301, the pattern returns the subjects of those triples for
`?r`, or John, Tom, Gary, and Bob. The problem with this is that Mary is also a reviewer by
20 virtue of rule 301; consequently, when the rule is taken into account, the triples include not only
those with the `ReviewerOf` predicate, but those that have the `ChairpersonOf` predicate, and
that adds Mary to the list of subjects for `?r`. An RDF model 101 and the rules and other
information required to interpret the model are termed together in the following an *RDF dataset*
Components of an RDF data set are shown at 313 in FIG. 3. The components include RDF model
25 101, with its schema 111 and RDF data 113, one or more optional rulebases containing rules
relevant to the model, and a list of optional aliases 323, which relate names used in the model to
longer designations.

The rulebases include an RDFS rulebase 319 which is a set of rules which apply to all RDF models.
30 An example of the rules in this rulebase is the rule that states that an entity which belongs to a
subclass of a class also belongs to the class, for example, that as a member of the class

Ph.D. Student, John is also a member of the class Person. In addition, rules may be defined for a particular RDF model. Rule 301 is an example of such a rule. These rules are contained in one or more other rule bases 321. Aliases 323 relates short names used in a model to the URIs that completely identify the short name. For example, John, Mary, Tom, Gary, and Bob are all
5 subjects and must therefore be identified by URIs. Aliases 323 will include a table that relates each name to its corresponding URI.

Systems for querying RDF models

A number of query languages have been developed for querying RDF models. Among them are:

- 10 • RDQL, see *RDQL - A Query Language for RDF*, W3C Member Submission 9 January 2004, <http://www.w3.org/Submission/2004/SUBM-RDQL-20040109>;
 - RDFQL, see *RDFQL Database Command Reference*, <http://www.intellidimension.com/default.rsp?topic=/pages/rdfgateway/reference/db/default.rsp>;
 - 15 • RQL, see G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, M. Scholl. RQL: A Declarative Query Language for RDF. *WWW2002*, May 7-11, 2002, Honolulu, Hawaii, USA.
 - SPARQL, see *SPARQL Query Language for RDF*, W3C Working Draft, 12 October 2004, <http://www.w3.org/TR/2004/WD-rdf-sparql-query-20041012/>.
 - SquishQL, see *RDF Primer*. W3C Recommendation, 10 February 2004, <http://www.w3.org/TR/rdf-primer>.
- 20 The query languages described in the above references are declarative query languages with quite a few similarities to SQL, which is the query language used in standard relational database management systems. Indeed, systems using these query languages are typically implemented on top of relational database systems. However, because these systems are not standard relational database systems, they cannot take advantage of the decades of engineering that have been invested
25 in the standard relational database systems. Examples of the fruits of this engineering that are available in standard relational database systems are automatic optimization, facilities for the creation and automatic maintenance of materialized views and of indexes, and the automatic use of available materialized views and indexes by the optimizer. What is needed if RDF triples are to reach their full potential are a technique for using RDF patterns to query sets of RDF triples that
30 may be employed in a standard relational data base management system and techniques for using the facilities of the relational database management system to reduce the cost in processing time of queries on sets of RDF triples. Providing such techniques is an object of the present invention.

Summary of the invention

The object of the invention is attained not only for RDF patterns but for queries written in other non-SQL languages as well. The techniques of the invention involve the TABLE function mechanism that is a standard component of most relational database systems. In one aspect, a non-SQL query is integrated into a relational database management system by including a table function invocation which includes a parameter that specifies the non-SQL query in an SQL query. Query execution code is associated with the table function. When the SQL query is executed, the query execution code executes the non-SQL query on data that is accessible to the table function and returns results of the execution of the non-SQL query to the SQL query.

In another aspect, SQL query string generating code is associated with the table function. When the SQL query string generating code is executed, it creates an SQL query string that is equivalent to the non-SQL query. The relational database management system executes the SQL query string generating code and replaces the table function invocation in the containing query with the generated SQL query string prior to executing the containing query. A table function may have both the query execution code and the SQL query string generating code associated with it or it may have only the one or the other.

One use of the techniques of the invention is to integrate RDF data into a relational database system so that queries on sets of RDF triples that use RDF patterns may be done in a relational database system. In this use, the table function invocation's parameter is an RDF pattern.

In other aspects of this use, the table function invocation's parameters may additionally include a specification of an RDF model that contains a set of RDF triples and a specification of an RDF rulebase. When an RDF model is specified, the RDF pattern is applied to the specified RDF model. When an RDF rulebase is specified, the rulebase is applied to the RDF model to infer additional RDF triples and the RDF pattern is applied to the inferred triples as well as to the triples in the RDF model.

Additional aspects of using the technique to integrate RDF data into a relational database system include applying optimizations available in the relational database system to the query performed by the TABLE function. The RDF triples accessed by the TABLE function are stored in an RDF triples table, and one of the optimizations is an index on the RDF triples table. The queries on the

RDF triples table include self joins and another optimization is materialized views of the self joins. A further optimization is a subject-property matrix join materialized view in which a single row in the materialized view will have fields for a subject and for a number of different predicates relevant for the subject. Each of the predicate fields will have a value of an object in an RDF triple which

5 has the kind of predicate specified in the column and for which the row's subject is the subject. This optimization may be used in any situation where a table has separate records for different attributes of a particular entity. Yet another optimization is inferring RDF triples by applying an RDF rule base to an RDF model , making a table of inferred RDF triples for a model, and using the table of inferred RDF triples in addition to the model when an RDF pattern is applied to the model.

10 Other objects and advantages will be apparent to those skilled in the arts to which the invention pertains upon perusal of the following *Detailed Description* and drawing, wherein:

Brief description of the drawing

- FIG. 1** shows RDF triples represented as graphs;
- 15 **FIG. 2** shows tables of RDF triples and an RDF pattern;
- FIG. 3** shows an RDF rule and RDF information;
- FIG. 4** provides an overview of a relational database management system in which the invention is implemented;
- FIG. 5** shows an SQL query that contains an RDF_MATCH table function;
- 20 **FIG. 6** shows RDF triple tables 445 in a preferred embodiment;
- FIG. 7** is an illustration of an inferred triple index and the API for manipulating such indexes;
- FIG. 8** is a flowchart of the operation of the RDF_MATCH table function;
- FIG. 9** is an illustration of self joins;
- FIG. 10** is an illustration of RDF_MATCH optimization tables 447;
- 25 **FIG. 11** is an illustration of a subject-property matrix join materialized view;
- FIG. 12** is an illustration of a predicate, subject, object index;
- FIG. 13** shows RDF patterns used to determine the efficiency of various kinds of indexes;
- FIG. 14** shows how rewriting the contained query can improve efficiency;
- FIG. 15** is an example of rewriting;
- 30 **FIG. 16** is a flowchart of optimization by rewriting;
- FIG. 17** shows RDF rule tables 449 in a preferred environment;
- FIG. 18** shows the API for manipulating rulebases in a preferred environment;

FIG. 19 shows the API for manipulating materialized views in a preferred environment; and **FIG. 20** shows further examples of the use of `ODCITableRewrite`.

- 5 Reference numbers in the drawing have three or more digits: the two right-hand digits are reference numbers in the drawing indicated by the remaining digits. Thus, an item with the reference number 203 first appears as item 203 in FIG. 2.

Detailed Description

- 10 The following *Detailed description* will first present an overview of the invention as embodied in a standard relational database management system (RDBMS) and will then present details of a preferred embodiment.

Overview of the invention

- 15 *Overview of a RDBMS in which the invention is implemented: FIG. 4*

FIG 4 is a functional block diagram of a relational database management system 401 in which the invention is implemented. RDBMS's are characterized by the fact that the information they contain is organized into tables having rows and named columns. A row of data establishes a relationship between the items of data in the row and the SQL query language uses the relationships thus
20 established to locate information in the tables. RDBMS system 401 may be any relational database system which employs a variant of the SQL language that includes table functions. As will be explained in more detail in the following, a table function is a function which permits the RDBMS system to treat a collection of data that is obtained by the function as a table.

- 25 The main components of RDBMS system 401 are a processor 421, memory 403, which contains data and programs accessible to the processor, and persistent storage 423, which contains the information organized by system 401. Processor 421 further can provide information to and receive information from display and input devices 422, can provide information to and receive information from networks 424, and can provide information to and receive information from file system 426.
30 RDBMS system 401 is created by processor 421 as it executes programs in memory 403 using data contained in memory. The programs typically include an operating system 407, which manages the resources used by RDBMS 401, relational database program 409, which interprets the SQL

language, and application programs 411, which provide queries to RDB program 409. Data used by these programs includes operating system data 419, used by the operating system RDBMS data 417, used by RDB program 409, and application program data 415, used by application programs 411.

- 5 The information which RDB program 409 maintains in persistent storage 423 is stored as *objects* that RDBMS system 401 is able to manipulate. Among the objects are fields, rows, and columns in the tables, the tables themselves, indexes to the tables, and functions written in the SQL language. The objects fall into two broad classes: user-defined objects 441, which are defined by users of the RDBMS, and system-defined objects 425, which are defined by the system. RDBMS 401 maintains
- 10 definitions of all of the objects in the database system in data dictionary 427, which is part of DB system objects 425. For the present discussion, the most important definitions in data dictionary 427 are table definitions 429, which include definitions 431 of RDF tables 443, table function definitions 433, which define table functions including RDF_ MATCH table function 435, which permits use of RDF patterns to query RDF models in RDBMS 401, and SQL function definitions
- 15 437, which includes RDF_GENMODEL function 439, which takes RDF triples and makes them into RDF tables 443.

The tables of interest in user objects 441 are RDF tables 443, which are tables in RDBMS 401 that are made from the information contained in RDF information 313. These tables fall into three

20 groups: RDF triple tables 445, which represent the triples making up an RDF model 101, RDF rule tables 449, which contain the rule bases belonging to RDF information 313, and RDF optimization objects 447, which are tables and other objects which are used to speed up queries on the RDF models represented by RDF triple tables 445 and the RDF rules in rules tables 449. All of these tables and objects will be explained in more detail below.

25

Overview of the operation of the invention: FIG. 5

The invention integrates RDF into SQL by means of a set of tables 445 and 449 in user objects 441 that represent RDF data sets and a table function RDF_MATCH that takes a specification of an RDF data set and an RDF pattern as parameters and returns a set of result

30 rows of triples from the RDF data set that match the RDF pattern.

The solution of the RDF pattern may include inferencing based on RDFS and user-defined rules. The signature of RDF_MATCH is as follows:

```

RDF_MATCH (
  Pattern          VARCHAR,
  Models           RDFModels,
  RuleBases        RDFRules,
5  Aliases          RDFAliases,
)
RETURNS AnyDataSet;

```

The first parameter is one or more character strings that indicate the RDF pattern to be used for the query. Typically, the character string will consist of one or more <Subject, Property, Object> triple patterns. The remaining parameters specify the RDF data set to be queried. `Models` specifies the data set's RDF models, `RuleBases` specifies rule bases that contain the RDF rules that apply to the models, and `Aliases` specify any aliases that apply to the RDF data set. As is true with any table function, `RDF_MATCH` returns a set of result rows. Each result row represents a triple

consisting of a set of values (bindings) for the variables used in the pattern. Substitution of the variables in the pattern with the corresponding values will result in an RDF graph that is a subgraph of the graph represented by the RDF dataset (including rulebases) against which the query has been posed.

It should be noted that the contents of the result rows returned by `RDF_MATCH` will depend on the RDF pattern used in the query and the RDF data against which the query is run. For this reason, the return type for `RDF_MATCH` has been defined as `AnyDataSet`, which is a collection of tuples of a generic type called `AnyData`. When an SQL query employs the `RDF_MATCH` table function, components of the query such as its `SELECT`, `WHERE`, `ORDER BY`, etc., clauses can reference the variables present in the RDF pattern simply by the variable names.

FIG. 5 shows an example RDF query using the `RDF_MATCH` table function at 501. Query 501 returns information from RDF model 101 about student reviewers who are less than 25 years old. The query employs an SQL `SELECT` statement 503. In very general terms, a `SELECT` statement

selects field values from one or more database tables into result rows produced by the `SELECT` statement. A filter in the `SELECT` statement may determine what result rows are selected and the `SELECT` statement may also specify operations to be performed on the result rows. When the `SELECT` statement includes a table function 505, the table function provides the data for the result rows.

Continuing in more detail, the SELECT statement specifies at 512 that the result rows will be selected from a table *t* which is provided by the table function. At 504 is specified the relationship between the columns *t.r*, *t.c*, and *t.a* of table *t* and the variables *?r*, *?c*, and *?a* of RDF pattern 507. Thus, in each record of table *t*, a field for the *t.r* column will contain a value of *?r*, and so on for the other fields. At 505, the SELECT statement specifies that the rows will be provided by the *RDF_MATCH* table function. In the following, the query that contains a table function will be called the *containing query* for the table function.

The parameters of *RDF_MATCH* table function 505 include the RDF pattern 507 which will be used to select information from RDF model 101. In conjunction with the relationship specified at 504, the pattern assigns the person selected for each row of *t* to the *t.r* column, the conference selected for each row of *t* to the *t.c* column and the person's age to the *t.a* column. As required by RDF pattern 507, the persons who will have rows in table *t* will be persons who are students and reviewers for any of the conferences indicated in RDF model 101. The remaining parameters are the following:

- *RDFMODELS* 509 specifies the RDF model the query is being applied to.
- *NULL* 511 that no rule base is to be included;
- *NULL* 513 indicates that no aliases are involved.

When SELECT statement 503 executes on RDF data triples 203, *RDF_MATCH* 505 returns rows 516 which contain the information from the *?r*, *?c*, and *?a* fields belonging to the RDF triples that match RDF pattern 507. In this case, there is a row for each student reviewer-conference combination and each of the rows contains the student's age. In addition to the values for the RDF pattern's variables, the result rows indicate the types of the object variables. That is necessary because objects may have either URI values or literal values. Thus, *?c* specifies conferences, which are specified by URI, so the result rows include the column *c\$type*, which indicates that the type of values of *c* is URI. In the case of *a*, the values are integer literal values, and *a\$type* indicates that fact.

SELECT statement 503 then selects fields belonging to the columns `r`, `c`, and `a` of rows 515 to produce result rows 514. The WHERE clause of SELECT statement 503, finally, limits the result rows produced by the execution of the SELECT statement to ones in which the age of the people selected by pattern 507 is less than 25. In this case, the WHERE clause has no effect, since all of the student reviewers are under 25. It should be noted here that because the RDF schema also consists of RDF triples, `RDF_MATCH` can also be used to query an RDF schema, for example, to obtain the domains and ranges for a property.

An advantage of using the `RDF_MATCH` table function in a SELECT statement to query RDF data is that any SQL construct that can be used with a SELECT statement can be used to further process the result rows 516 returned by `RDF_match`. These constructs include iterating over the result rows, aggregating values contained in the result rows, constraining the result rows using WHERE clause predicates, sorting the result rows using ORDER BY clauses, and limiting the result rows by using the ROWNUM clause. Also, the SQL set operations can be used to combine result sets of two or more invocations of `RDF_MATCH`. Support for OPTIONAL matching (as described in the SPARQL reference) can be provided using the OUTER JOIN operation in SQL.

An example of how the SQL COUNT and AVG constructs might be used in a SELECT statement that contains table function 505 is shown at 515. Query 515 specified by SELECT statement 517 uses the same FROM TABLE clause 507 as query 501 and consequently the same RDF model and the same RDF pattern, but the query returns a set of result rows that specify for each conference, the number of student reviewers for the conference and the average age for the conference. The returned result rows are shown at 523. Result rows 523 are made from information in the rows returned by RDF pattern 507. These rows are shown at 514. As specified at 518, the result rows contain three fields, one indicating the conference, one indicating the number of student reviewers, and one indicating the average age of the student reviewers. There are two rows, one for each conference. The number of student reviewers is computed using the COUNT function, which counts the number of rows for each conference in the rows returned by the RDF pattern and their average age is computed by the AVG function. The GROUP BY clause specifies that the results in result rows 523 are grouped by conference and the ORDER clause specifies that the results are ordered by the average age of the student reviewers.

A preferred embodiment of RDF_MATCH: FIGs. 6-8

The following discussion of a presently-preferred embodiment of RDF_MATCH will begin with an overview of how table functions are implemented, will disclose details of RDF triples tables 445 in a preferred embodiment, will then provide a detailed disclosure of RDF_MATCH's operation, and will finally discuss optimizations. The preferred embodiment is implemented using a relational database management system manufactured by Oracle Corporation, Redwood City, CA. A table function like RDF_MATCH can, however be implemented in any RDBMS that supports table functions. The optimizations may be implemented in any RDBMS that supports materialized join views and indexes. Details of the implementation of table functions used in the preferred embodiment may be found in *Data Cartridge Developer's Guide Release 2 (9.2)*, Part No. A96595-01, Oracle Corporation, March 2002, which is hereby incorporated by reference into the present patent application.

Implementation of table functions

Oracle relational database management systems provide users of the system with a standard interface for defining and implementing a table function. The interface includes three methods:

- OCIDTableStart, which does whatever initialization is required before the table function can return data;
- OCIDTableFetch, which performs whatever action is necessary to fetch the data returned by the table function; and
- OCIDTableClose, which does whatever cleanup is necessary after the table function has ceased returning data.

A user who is defining a table function must provide an implementation for each of these methods. RDF_MATCH is a built-in table function and implementations of the methods are provided by Oracle Corporation.

RDF triples tables 445: FIG. 6

FIG. 6 shows the RDF triples tables 445 in which the data for an RDF model 202 is stored after normalization. There are two main tables: IdTriples 601, which is a list of models and their RDF triples, as represented by internal identifiers for the URIs and literals making up the triple, and

UriMap 613, which maps URIs and literals to the internal identifiers and thus permits conversions between the URIs and literals and the internal identifiers. This arrangement saves storage space and increases efficiency by permitting the URIs, which are often lengthy, and the literals, which are also often lengthy and may further have a variety of types, to be replaced in IdTriples table 601 by
 5 internal identifiers having a single type and size.

Continuing in detail with IdTriples table 601, this table has a row 611 for every RDF triple in the RDF models that have been loaded into RDBMS 401 on which the RDF_MATCH function is being executed. The table has four columns:

- 10 • ModelID 603, which contains the internal identifier of the model to which the RDF triple belongs;
- SubjectID 605, which contains the internal identifier for the RDF triple's subject;
- PropertyID 607, which contains the internal identifier for the RDF triple's predicate; and
- ObjectID 609, which contains the internal identifier of the RDF triple's object.

15 As shown in FIG. 6, IdTriples table 601 shows the rows for the first four data triples of data triples 203. It would of course contain a row for every schema triple in table 201 and every data triple in table 203.

UriMap table 613 has a single row 619 for every internal identifier which appears in IdTriples
 20 table 601. There are four columns that are of interest in the present context:

- InternalID 615, which contains the internal ID; and
- RDFVal 617, which specifies a URI or literal value corresponding to the internal ID;
- a flag which indicates whether the value of RDFVal 617 is the canonical form for the value;
- the type of RDFVal 617.

25 Types include URIs, strings, and integers.

The canonical form for a value is a standard form for writing the value. For example, the numeric value 24 may be written as 024, 24.00, 2.4×10^1 , and so on. Depending on the application, any of these may be a canonical form. In a preferred embodiment, the form the value has when the first entry is made for the value in UriMap 613 is treated as the canonical value. There is further an
 30 index, idx_num 627, that indexes a given numerical value to a row in UriMap 613 that contains the canonical representation

RDF rules tables 449: FIGs. 17 and 18

The RDF rules that apply to an RDF model 101 are stored in RDF rules tables 449. There are two such tables: rulebase table 1701, which relates rules to the rule bases they belong to, and rule table 1709, which includes an entry for each RDF rule that has been input to system 401. Beginning with rule table 1709, the table has a row 1719 for each rule. Each row contains the rule's name in column 1711, the rule's left-hand RDF pattern in column 1713, a filter, which may be null, in column 1715, and the rule's right-hand RDF pattern in column 1717. The rule's name must be unique in table 1709. Rulebase table 1701 has a row 1707 for each rule that belongs to each rulebase that has been input to system 401. Column 1703 contains the name of a rulebase and column 1705 contains the name of a rule that belongs to that rulebase. The name of the rule in field 1703 is the rule's name from field 1711. As is apparent from this arrangement, a given rule may be part of many different rulebases.

Like models, rulebases may be received as XML files; in such a situation, entries for the rules are added to rule table 1709 and entries for the rules and rulebases to rulebase table 1701. There is further an application programmer's interface (API) for creating rulebases, deleting rulebases, and incorporating rulebases in other rulebases. FIG. 18 provides an overview of this rulebase API 1801. At 1803 and 1805 are shown the functions for creating and dropping rulebases; they take the name of the rulebase being created or dropped as parameters. The functions at 1807 and 1809 permit rules to be inserted into and deleted from a rulebase; the parameters are the name of the rulebase and the name of the rule..

The interface for creating a rule is shown at 1811; the result of its execution is a new entry in rule table 1709. The interface for dropping a rule from a rulebase is shown at 1813; the result of its execution is the removal of the entry for the specified rulebase and rule from rulebase table 1701.

Details of the operation of RDF_MATCH: Fig8

There are two stages in the operation of RDF_MATCH table function 435. The first stage occurs when RDBMS system 401 compiles the SELECT statement containing table function 435; the second stage occurs when the SELECT statement is executed.

Compile-time determination of the form of the rows to be returned by RDF_MATCH:

When the SELECT statement is compiled, it can be determined which columns of the result rows returned by REF_MATCH will be actually required for the result rows returned by the SELECT statement. The results of this determination are provided to RDF_MATCH, which can use it to optimize the queries it makes on RDF triples tables 445.

5

Execution time generation of the query performed by the RDF_MATCH table function: FIG. 8

At execution time, the RDF_MATCH table function uses the information in the RDF pattern contained in the table function to generate a query on RDF triple tables 445 that obtains result rows that have the form specified in the containing SELECT statement and the values specified by the variables in the RDF pattern. The query generated by RDF_MATCH will be termed in the following the *generated* query. In overview, the generated query includes the following:

10

- a subquery on UriMap 613 to convert the values of literals and URIs specified in the RDF pattern to their corresponding internal IDs.
- a subquery on IdTriples table 601 that uses the internal IDs to find the triples that satisfy pattern 507.
- another subquery on UriMap 613 converts the internal IDs in the results of the subquery on IdTriples table 601 to their corresponding URI and literal values.

15

The corresponding URI and literal values are then output as specified at 504 in the SELECT statement. The subquery on IdTriples table 601 involves *self joins*. A *join* is a query that combines rows from two or more tables. In a *self join*, the rows that are combined are from the same table.

20

FIG. 8 is a flowchart 801 which provides more detail of how RDF_MATCH generates and executes the query on RDF triple tables 445. In the preferred embodiment, the query is generated by the ODCITableStart method and executed by the OCDITableFetch method. At 803 is shown the call to the method, with the method's parameters of an RDF pattern, one or more RDF models, at least the RDFS rules, and in some cases, aliases. Query generation is shown at 805. The first step is to expand aliased URIs in the RDF patterns, so that all URIs in the RDF patterns have their full URI values (807). The next step is to generate queries on literal tables 623 and UriMap table 613 to convert the URIs and literals in the RDF pattern into their corresponding internal identifiers (809).

25

30

The next step is to generate a query on `IdTriples 601` that produces a result table containing the rows of `IdTriples 601` that satisfy each of the triples in the RDF pattern (810). For RDF pattern 507, the result table contains all of the rows having the `ReviewerOf` predicate, all of the rows having the predicate `rdf:type` and an object belonging to the class `Student`, and all of the rows having the predicate `Age`. The generated query that does this is shown at 823. Once the query for the result table has been generated, a self-join query can be generated on the result table which returns the set of rows from the result table for which the RDF pattern's variables match across the rows as specified in the RDF pattern (811). In the case of RDF pattern 507, the variable in question is the variable `?r`, which represents the subject in each of the triples of pattern 507. The query that is generated for pattern 507 is shown at 827. At step 813, a limitation is added to the generated query that limits the rows to those belonging to the model specified in the invocation of `RDF_MATCH`. In the present case, that is the model specified in the invocation with `'reviewers'`.

After the query on `IdTriples 601` has been generated, queries are generated on `UriMap` table 613 and literal tables 623 to convert the internal identifiers in the self-join query results to their corresponding URI and literal values (815). Finally, the RDF rules that are specified in the invocation of `RDF_MATCH` are taken into account by replacing references to `IdTriples` table 601 in the generated query with subqueries or table functions that yield not only the triples explicitly specified in the RDF pattern, but also the triples which may be inferred by applying the rules to the explicitly specified triples (817). Rule processing is explained in more detail in the following. Once the query has been generated as just described, relational database management system 401 applies its optimizers to the query (818) and then executes the optimized query (819). The results are output in rows that have the columns that were determined at compile time.

Creating RDF triple tables 445 and RDF rule tables 449

As already described, RDF data sets are generally represented as text in text files and in many cases, the text files are written in a dialect of XML that has been developed for representing RDF data sets. The RDF data set contained in a text file may be added to RDF triples tables 445 and RDF rules tables 449 by any technique which reads the text file and converts its contents into records in RDF triples tables 445 and RDF rules table 449. In general, conversion works as follows:

- extract an RDF triple from the text representation of an RDF model.
 - for each URI and literal in the triple, determine whether there is already a row in UriMap 613 for either the URI or the literal. If there is no row, a new row is made and the InternalID field for that row is the internal ID for the URI or literal.
- 5 • When the internal ID for each component of the triple has been obtained from UriMap, make an entry for the triple in IdTriples table 601.

With rules, when a rule is encountered in a text file, the text strings specifying the rule are written to rule table 1709.

- 10 A preferred embodiment of the invention provides a function called RDF_GENMODEL which can be used in the invocation of RDF_MATCH to specify the RDF model. RDF_GENMODEL's signature looks like this:

```
RDF_GENMODEL (Webpages RDFWebpages)
RETURNS VARCHAR;
```

15

The parameter is the URI for a Web page that contains the XML representation of an RDF model. An invocation of RDF_MATCH that uses RDF_GENMODEL looks like this:

```
20 SELECT t.a age
FROM TABLE(RDF_MATCH(
    '(?r age ?a)'
    RDFModels(RDF_GENMODEL(<web_page_uri>)),
    NULL,
    NULL)) t
25 WHERE t.a < 25
```

When RDF_MATCH is invoked at query execution time, RDF_GENMODEL is executed. The function reads the XML representation of the RDF model contained in the web page and makes non-persistent versions of RDF triples tables 445 whose entries correspond to the triples in the Web page. RDF_MATCH then applies the invocation's RDF pattern to the tables made by RDF_GENMODEL.

30

Rule processing

To handle rules, the RDF_MATCH function replaces references to IdTriples table 601 in the generated query with subqueries or table functions that yield not only the triples explicitly specified

35

in the RDF pattern, but also the triples which may be inferred by applying the rules to the explicitly specified triples. Rule table 1709 is queried to determine what subqueries and/or table functions are necessary to obtain the inferred triples from `IdTriples` table 601 and the subqueries and/or table functions are applied to `IdTriples` table 601. Taking RDF pattern 312 and rule 301 as an example, there is a row in rule table 1709 for rule 301. When the RDF query specified in pattern 312 is executed on a model which includes the rulebase `rb`, rule table 1709 is queried for rules whose right hand side triple specifies a person who is a `ReviewerOf` a conference; if any are found, the left hand side of the triple is used as well as the first triple of pattern 301 to select people to whom the remaining triples of pattern 301 are to be applied.

Subqueries are used whenever the required inferencing can be done conveniently within a SQL query (i.e., without explicitly materializing intermediate results). The inferencing for rule 312 is done in that fashion. These subqueries generally take the form of a SQL UNION with one UNION component for each rule that yields a triple that selects entities inferred by the rule, plus one component to select the triples explicitly specified in the query. Table functions are used when the subquery approach is not feasible.

Processing RDFS inference rules

The RDFS inference rules require computation of transitive closures for the two transitive RDFS properties: `rdfs:subClassOf` (rule `rdfs11`) and `rdfs:subPropertyOf` (rule `rdfs5`). In Oracle RDBMS, these transitive closures can be computed using hierarchical queries with the `START WITH` and `CONNECT BY NOCYCLE` clauses. Note that `CONNECT BY NOCYCLE` queries can handle graphs that contain cycles. The remaining RDFS rules can be implemented with simple SQL queries.

To ensure that RDFS inferencing can be done within a single SQL query, the user is prohibited from extending the built-in RDFS vocabulary. This means, for example, that there cannot be a property that is a sub-property of the `rdfs:subPropertyOf` property, nor can there be a user-defined rule that yields `rdfs:domain` triples.

Processing user-defined rules

User-defined rules can be classified as follows based upon the extent of recursion, if any, in the rule:

- Non-recursive rules: The antecedents cannot be inferred by the given rule, or any rule that depends on the given rule's consequents.
- 5 • Simple recursive rules: These rules are used to associate transitivity and symmetry characteristics with user-defined properties.
- Rules that use arbitrary recursion unlike the other two categories.

Non-recursive user-defined rules can be evaluated using SQL (join) queries by formulating the FROM and WHERE clauses based upon the antecedents and the SELECT clause based on the
10 consequents of the rule so as to return the inferred triples. Note that the triples that match the antecedents of a user-defined rule could themselves be inferred, so the FROM clause may reference subqueries to find inferred triples.

Simple recursive rules involving transitivity and symmetry can be evaluated as follows. Symmetry
15 can be easily handled with a simple SQL query. However, handling transitivity with a single SQL query requires some type of hierarchical query (e.g., using the START WITH and CONNECT BY NOCYCLE clauses in Oracle RDBMS), as in the case of transitive RDFS rules.

The third class of rules involving arbitrary recursion is the most complicated, and it has not been
20 addressed in the presently-preferred embodiment. Because an unknown number of passes over the intermediate results is required to find all inferred triples, these rules must be evaluated using table functions.

Optimizations of RDF_MATCH: FIGs. 10-16, 19-20

25 A number of optimizations of RDF_MATCH are possible. The optimizations fall into two categories:

- adding objects to RDF tables that increase the speed of execution of the queries specified in the RDF patterns and
- preprocessing RDF_MATCH to produce a set of declarative SQL strings that specify a query or
30 queries that are equivalent to the query generated by RDF_MATCH and rewriting the containing query by replacing the TABLE construct and RDF_MATCH invocation with the SQL strings.

Optimization by preprocessing a table function and rewriting the containing query using the generated query can be applied to any table function for which a set of declarative SQL strings can be generated at compile time for the containing query that is completely equivalent to the query generated by the table function at runtime. The above is the case when nothing occurs during execution of the table function that will affect the form of the result rows returned by the table function.

RDF Optimization tables 447: FIG. 10

Among the ways in which `RDF_MATCH` can be optimized by adding tables to RDF optimization tables 447 are the following, shown at 1001:

- Generating materialized join views to reduce the join cost (1001);
- Generating subject-property matrix materialized views (1003);
- Generating indexes for the `IdTriples` table (1005); and
- Adding models with inferred triples to the `IdTriples` table (1009).

Materialized views

A SQL query operates on one or more named tables to produce result rows. If the result rows are given names in the SQL query, then the query can operate on them in the same fashion as it can on any other table. The ultimate source of all the data in an SQL query is one or more *base tables* which are always present in DBS persistent storage 423. In the present context, the base tables are `IdTriples` 601 and `UriMap` 613. The named tables made from result rows are termed *views* of the base tables. For example, the query fragment 823 produces three views of the base table `IdTriples`: `t1`, which contains the rows of `IdTriples` whose `PropertyID` indicates the predicate `Reviewers`, `t2`, which contains the rows whose `PropertyID` indicates the predicate `rdf:type` and the `ObjectID` `Student`, and `t3`, which contains the rows whose `PropertyID` indicates the predicate `Age`. The relational database management system creates the views when the query is executed and removes them when they are no longer needed for the execution of the query. Creating the views takes a considerable amount of processing time, and consequently, the speed with which a query can be executed can be increased by the use of *materialized views*. A materialized view is simply a persistent view, that is, one that exists prior to the execution of the query and remains in existence after its execution. The costs of a materialized

view are of course the extra persistent storage that it requires and the costs associated with keeping the data in the materialized view consistent with the data in the base tables it is a view of.

Note that the creation of materialized views does not complicate the logic of RDF_MATCH implementation. That is, the basic scheme of generating a self-join query as described above is still applicable. The only difference is that the RDBMS cost-based optimizer optimizes the generated self-join query by rewriting it to use materialized views where they are available and their use reduces the cost of the query in terms of I/O and CPU time.

10 Generic materialized join views: FIG. 9 and FIG. 19

If the same variable is used in more than one triple of the search pattern, the query generated by RDF_MATCH table function 435 involves a self-join of the IdTriples table, as may be seen from the FROM clause IdTriples t1, IdTriples t2, IdTriples t3 at 823. Depending on how many triples patterns are specified in the RDF pattern, a multi-way join needs to be executed.

15 Since the join cost is a major portion of the total processing time, materialized join views can be defined to speed up RDF_MATCH processing. The row size of IdTriples table 601 is small and hence the trade off between the additional storage space required for materialized views and the extra processing speed they provide favors the use of materialized views. In general, six two-way joins may be defined on IdTriples table 601, namely joins between SubjectID-
20 SubjectID, SubjectID-PropertyID, SubjectID-ObjectID, PropertyID-PropertyID, PropertyID-ObjectID, and ObjectID-ObjectID. Examples of some of these joins are shown at 901 in FIG. 9. In each case, a concrete example 903-913 of the join is given along with the number of rows in data triples table 203 which will be rows of the self-join:

25 Which of these six self joins is worth being treated as a materialized view depends on the kinds of RDF patterns that can be expected to occur in the RDF_MATCH function. Selection of the joins to be made into materialized views can thus be based on the workload characteristics. The most common joins are typically SubjectID-SubjectID, SubjectID-ObjectID, and ObjectID-ObjectID. Database management system 401 incrementally maintains the
30 materialized join views to keep them current with IdTriples table 601 whenever they are used in a query.

The API for generic materialized views: FIG. 19

FIG. 19 shows the API 1901 used to create and maintain generic materialized views. The first function in the API, `RDFMViewCardinalities` 1903, takes the name of an RDF model and optionally the name of an RDF rulebase as parameters. For the RDF model, the function analyzes the rows of RDF triples belonging to the RDF model in `IdTriples` 601s and the additional triples inferred by the rulebase specified in the rulebase parameter and generates cardinalities of materialized join views between Subject-Subject, Subject-Object, Subject-Property, Property-Property, Property-Object, and Object-Object so that a user can estimate the size of the join views to decide whether or not he/she wants to create the join views.

`RDFMviewCreate` function 1905 creates a generic materialized view based on the RDF models and optional rule bases specified in the second and third parameters. The materialized view will contain the model's triples and the triples inferred from the model using the rule bases. The first parameter is the name of the materialized view to be created. The fourth parameter specifies the join columns for the materialized view: SS indicates that the subject columns for the triples are the join columns; SO specifies the subject and object columns; SP the subject and predicate columns; PP the predicate columns, PO the predicate and object columns; and OO the object columns. `RDFMViewDrop` function 1907 deletes a named materialized view.

Subject-predicate matrix materialized views: FIGs. 11 and 19

RDF triples are extremely expressive in the sense that just about any fact can be expressed using an RDF triple. A table of RDF triples is, however, not ideal for efficient query processing. For example, data triples table 203 contains a separate row for each of a subject's predicate-object combinations. John, for instance, has rows for his age, for his membership in the class of Ph.D students, and his function as a reviewer for ICDE 2005. Because this is the case, obtaining all of the information about John from the table requires a three-way self join. Indeed, if John has n predicate-object combinations, retrieving all of the information about John from the table requires an n -way self join.

Query performance can be improved significantly by creating a *subject-predicate matrix*, that is, a materialized join view in which a row for a particular subject contains a number of different objects

to which the particular subject is related. The objects may be either *directly* or *indirectly* related to the particular subject. Directly related objects are objects that belong to RDF triples that have the particular subject. Indirectly related objects are objects that belong to RDF triples whose subjects are objects in RDF triples which have the particular subject. There may of course be more than one
5 level of such indirection. The columns of the subject-property matrix include a column for the subject and a column for each of the kinds of object related to the subject in the row. FIG. 11 shows at 1101 how a subject-predicate matrix 1105 may be made from a table of RDF triples 1103. The subjects of subject-predicate matrix 1105 are limited to Ph.D students. For such subjects, table 1105 has columns for the subject and columns for kinds of objects that are related to the students in
10 table 1103, namely Age objects and StudiesAt objects. There is a row in subject-predicate table 1105 for each of the subjects in RDF triples table 1103 that belong to the class Ph.D student. In table 105, the Age objects are directly related to the subjects John and Pam and the StudiesAt objects are indirectly related to those subjects via the Univ1 and Univ2 objects and subjects in table 1103.

15 The subject-predicate matrix can be used to process RDF queries efficiently. For example, consider RDF pattern 1107, which retrieves the Age and EnrolledAt objects for each student belonging to the class Ph.D. Student and the City objects for the universities that are the Enrolled At objects. Absent materialized view 1105, this query will require a 4-way self-join
20 on the IdTriples table (leaving out the conversion between Ids and URIs, for simplicity). However, by using the materialized view 1105, the query can be processed by simply selecting all the rows from the materialized view. Thus, self-joins can be completely eliminated in this case. This can lead to significant speed-up in query processing.

25 While subject-predicate matrix materialized views are particularly useful with tables of RDF triples, they may be used in any situation in which self joins are required to collect information about a number of attributes of a set of entities in a table. A query requiring an n -way self join to obtain the information about the attributes could potentially be processed using a matrix with columns for m -attributes using $(n - m)$ joins. Such matrices are most efficient in their use of
30 storage if each subject in the matrix has one or more objects for each of the chosen predicates. Some sparseness may be permitted to allow expanding the group of subjects to include subjects that may have no objects for a few of the predicates that have columns in the matrix. It may be noted

that as with materialized views generally, the performance gains from the use of such matrices must be traded off against the extra space required.

The API for making a subject-property materialized view is shown at 1909 in FIG. 19. The function
5 takes five parameters: a name for the new materialized view, the RDF model or models to which it is to apply, the rulebase or rulebases from which RDF triples may be inferred from the model, an RDF pattern 205 whose predicates are to be columns in the materialized view, and a filter for the table's rows.

10 *Indexing IdTriples table 601: FIGs. 12 and 13*

A common way of speeding up access to information in an RDBMS table is to provide an *index* for the table. Indexes on tables work exactly the same way as indexes in books. An index entry in a book has a word or phrase followed by a list of the numbers of the pages in the book on which the word or phrase occurs. The index in the book speeds access by permitting the reader to go directly
15 to the page or pages of interest. Conceptually, an index entry for an RDBMS table consists of a value from a row of the table followed by a row number for the row that contains the value. The value may be made by concatenating several fields of the row. The index on the table speeds access to the table by permitting the RDBMS to go directly to the indexed row or rows. RDBMS systems typically provide built-in facilities for creating a number of different kinds of indexes. The
20 kind of indexes used in the preferred embodiment are the B tree indexes provided by the Oracle database system in which the preferred embodiment is implemented.

As was the case with materialized views, the RDBMS's optimizer automatically determines whether there are indexes to a table being queried and if so, whether using one of the index will reduce the
25 processing time required for the query in question. An important metric used by the optimizer is *selectivity*. Selectivity refers to the percentage of rows in a table that are returned by a query. A query is considered highly selective if it returns a very limited number of rows. A query is considered to have low selectivity if it returns a high percentage of rows. The more selective a query is, the greater the cost savings from using an index.

30 FIG. 12 shows at 1201 data triples table 203 in which each row has been assigned a row number 1203. Index 1205 is an index for data triples table 203 which indexes the table according to values

from each row that are made by concatenating table 203's Predicate, Subject, and Object fields in the row. Conceptually, index 1205 may be seen as a table with a Predicate column 1207, a Subject column 1209, an Object column 1211, and a Rownum column 1213. Each row 1215 of index 1205 is an index entry. The entry 1215 contains the values of the Predicate, Subject, and Object fields of a single row of data triples table 203 and the value of RowNum in entry 1215 is the row number of the single row in table 203. Of course, in some cases, there may be more than one row of the indexed table that corresponds to an index entry 1215, and in that case, there will be a list of row numbers at 1213. In index 1205, the index entry includes values from the Predicate, Subject, and Object fields in that order, and in the terminology used in the following, index 1205 is a (predicate, subject, object) index.

As mentioned earlier, most of the work involved in executing the query generated by RDF_MATCH is performing self joins on IdTriples table 601. Since the self joins involve repeatedly referencing the rows of IdTriples table 601, having indexes that are adapted to the kinds of queries generated by RDF_MATCH is critical for the performance of RDF_MATCH. In the following, the self-join queries generated by RDF_MATCH are analyzed to determine which columns of IdTriples table 601 should be indexed for optimal query performance. The analysis is performed using the information 1301 in FIG. 13. There are typically two types of RDF patterns:

1. those in which, for a given predicate, subject is joined with subject, or object with object, as shown in RDF pattern 1303, which joins rows whose subjects belong to the domain of the ReviewerOf predicate with rows whose subjects belong to the domain of the Age predicate.
2. those in which for a given predicate, subject is joined with object, as shown at 1305, in which joins rows whose objects belong to the range of the ReviewerOf predicate with rows whose subjects belong to the domain of the rdf:type predicate.

The same query patterns can be observed as more triples are added.

Since IdTriples 601 only has three columns, only the following five kinds of indexes for predicates can be built on the table:

1. (PropertyID),

2. (PropertyID, SubjectID),
3. (PropertyID, SubjectID, ObjectID).
4. (PropertyID, ObjectID)
5. (PropertyID, ObjectID, SubjectID)

5 Index (1) above will be termed in the following a *single-column* index; indexes (2) and (4) will be termed *two-column* indexes; indexes (3) and (5) will be termed *three-column* indexes. With queries that returns less than 20% of the records in IdTriples 601, and are therefore highly selective, indexes such as (3) and (5) above, which use the values of all three columns in a row's index entry, have been found to be most efficient.

10

Optimizing inferencing: FIGs. 7 and 10

Rulebases specified in the RDF_MATCH table function's parameters are applied, by default, during query processing to the specified list of RDF models. However, if a rulebase is used frequently, then
 15 a new model containing the RDF triples inferred from one or more rule bases can be added to IdTriples table 601. The new model can then be used to speed up query execution. This is shown at 1009 in FIG. 10, where model A 1011 in IdTriples table 601 has triples 1013. To optimize execution of RDF queries on model A, a new model A_inferred 1015 has been added to IdTriples table 601. The new model 1015 includes the triples 1017 that have been
 20 inferred by the application of one or more rulebases to triples 1013 of model A. To take advantage of A_inferred model 1015, the Models parameter for the invocation of the RDF_MATCH table function includes both model A 1011 and model A_inferred 1015 but the RuleBases parameter does not include the rulebase or rulebases that were used to produce A_inferred 1015. When this is done, RDF_MATCH simply uses the triples in the models A
 25 and A_inferred rather than using the rulebases to infer the inferred triples from model A.

In other embodiments, inclusion of a set of inferred triples may be transparent to the user. In such an embodiment, the inferred triples can be stored in a separate table in which they are related to the model the triples are inferred from and the rulebase used to infer them. When an invocation of
 30 RDF_MATCH specifies a rulebase, the code for the function checks whether there are inferred triples for the model and rulebase specified in the invocation, and if there are, the code does not

again infer the triples, but instead joins the inferred triples from the rows for the model and rulebase in the inferred triples table to the triples from the model.

FIG. 7 shows the API used to add models with inferred rules to `IdTriples` table 601. The API used to add a model with inferred rules is shown at 711; it takes as parameters the model to which the rulebase is to be applied, the rulebase, and a name for the new model that will contain the triples inferred by applying the rules specified by the rule base to the specified model. The result of executing `CreateRulesIndex` is that a model 1015 will be added to `IdTriples` table 601 that has the name indicated in the last parameter and the additional triples that are inferred by applying the rule base of the second parameter to the model specified in the first parameter. The API used to drop a model containing inferred rules is shown at 713; the only parameter is the name of the index to be dropped.

15 *Eliminating the overhead of the table function: FIGs. 14-16*

The SQL table function mechanism is a general purpose mechanism for converting data that is accessible to the table function into rows. The mechanism works not only with table functions that obtain their data from existing relational tables, as is the case with `RDF_MATCH`, but also with table functions that read their data from files, fetch the data across the World Wide Web, or even receive feeds of data such as stock price information. One consequence of the generality of the table function mechanism is substantial overhead. For example, the time t_{total} required for processing an RDF pattern using `RDF_MATCH` table function 435 has the following components:

$$t_{\text{total}} = t_{\text{core}} + t_{\text{sql2proc}} + t_{\text{proc2canonical}} + t_{\text{canonical2sql}}$$

Here t_{core} represents the core processing time, that is, the cost of executing the SQL query that is generated by `RDF_MATCH` and performs the self-joins on `IdTriples` table 601 and any additional joins on `UriMap` table 613. Once the result rows of the generated query have been computed, the table function mechanism copies the rows into variables of `RDF_MATCH` (t_{sql2proc}) and then converts the values of these variables to a canonical format ($t_{\text{proc2canonical}}$) for the table function mechanism so that the mechanism can return the values to the containing query. When the mechanism returns the values in the canonical format to the containing query, it transforms them back into rows ($t_{\text{canonical2sql}}$).

The processing time represented by $t_{\text{total}} - t_{\text{core}}$ depends on the size of the result set returned by the table function to the table function mechanism and hence $t_{\text{total}} - t_{\text{core}}$ will dominate the cost of executing the table function when the table function result set size is large. This is shown in graph 1401 in FIG. 14. Graph 1401 shows how t_{total} in seconds for RDF_MATCH increases with the number of result rows returned by the query generated by RDF_MATCH. Graph 401 1401 further shows how t_{core} 1403, t_{sql2proc} 1405, $t_{\text{proc2canonical}}$ and $t_{\text{canonical2sql}}$ 1407, and other 1409 make up t_{total} .

As is apparent from graph 1401, eliminating the conversion overhead of t_{sql2proc} , $t_{\text{proc2canonical}}$, and $t_{\text{canonical2sql}}$ would enormously reduce the amount of time required to execute RDF_MATCH where RDF_MATCH returns a significant number of rows. That it should be possible to eliminate the conversion overhead can be seen from the fact that with RDF_MATCH, the conversions are performed on result rows from relational database tables, that is, on data that is *already* in the form required for the containing query. What the conversions do is convert result rows returned by the generated query to values of variables in RDF_MATCH, convert the variable values to the table function mechanism's canonical form, and then convert the values in the canonical form back into result rows that have the same form as the ones returned by the generated query.

In the case of table functions like RDF_MATCH, in which the table function obtains its data by means of a query on a set of relational tables, the conversion overhead can be eliminated by rewriting the query containing the table function such that the query used by the table function to obtain the data replaces the table function in the containing query. This is shown in FIG. 15. At 1501 is shown a query that employs the table function RDF_MATCH (1503). The RDF pattern used in the table function is shown at 1505. At 1507 is shown the query which RDF_MATCH generates from RDF pattern 1505; at 1509 is shown rewritten query 1501 in which table function 1503 has been replaced by generated query 1507.

The query rewrite of FIG. 15 can of course always be done by hand; however, the table function mechanism can be modified to rewrite the query containing the table function, and consequently, a version of the table function mechanism can be created which does the following:

- given a table function, use the table function to generate a query;

- return the query generated by the table function;
 - rewrite the containing query so that the generated query replaces the TABLE construct and the table function invocation in the containing query;
 - reparse the rewritten containing query; and
- 5 • execute the rewritten containing query.

In the following, this new version of the table function mechanism will be termed the *table function rewrite version*. The table function rewrite version may be used in any situation where the rows returned by the table function may be declaratively defined, as is the case where they can be defined by an SQL query.

- 10 In a preferred embodiment, the table function rewrite version of the table function mechanism is implemented by adding an `ODCITableRewrite` method to the definition of the table function. The method defines how the table functions parameters are to be used to generate an SQL query that can replace the TABLE construct and the table function in the containing SELECT statement. When the containing query is being compiled by the SQL compiler, the compiler executes the
- 15 `ODCITableRewrite` method to obtain an SQL query that is equivalent to the table function. The compiler then replaces the TABLE construct and the `RDF_MATCH` invocation with the SQL query. At 1415 is shown an example of a query with `RDF_MATCH` invocation 1503. The rewritten query which results is shown at 1419. In query 1419, the TABLE construct and the invocation of `RDF_MATCH` have been replaced by query 1507 generated by the `ODCITableRewrite` method.

- 20 In addition to using the `ODCITableRewrite` method to generate the query required to rewrite the containing query, the table function rewrite version of the table function mechanism must perform additional type checking to ensure that the columns referenced in the containing query are indeed returned from the generated SQL query as well as to ensure that the data types for columns
- 25 referenced in the outer query are compatible with the source datatypes in the generated SQL query. The additional type checking overhead required is, however, small, and unlike the conversion overhead of the present table function mechanism, does not increase with the size of the result set returned by the table function. The exact mechanism used to obtain the SQL string is of course immaterial; it may be, as above, a method associated with the table function or it may be a function
- 30 that takes the table function as a parameter.

FIG. 16 is a flowchart 1601 of how the table function rewrite version of the table function mechanism operates in a preferred embodiment. The processing is done during the compilation phase 1603 of SQL query processing. At 1604, compilation continues until the compilation is finished (1606). If a table function is encountered (1605), the compiler first determines whether the table function has a rewrite method (1611). If it does not, the table function is processed in the usual manner (1608) and compilation continues (1606). If the table function does have a rewrite method, the rewrite method is executed (1615). If the result of the execution is not an SQL string, the table function is processed in the usual manner and compilation continues (1608, 1606). If the result is an SQL string, the compiler replaces the table function invocation and the TABLE construct with the returned SQL string (1617). The compilation phase then reparses the rewritten query (1619). Compilation of the containing query then continues until it is finished.

Then the optimization stage of query generation is entered and optimization is done on the containing query as rewritten. An important advantage of rewriting the containing query with SQL that is equivalent to the table function is that the equivalent SQL is available to the optimizer. With standard table functions, the optimizer can optimize the queries generated by the table function and can optimize the containing query, but when the optimizer is optimizing the containing query, it must treat the table function as a “black box” and cannot take the queries generated by the table function into account. Once optimization is done, the containing query is executed with the replacement string. Here, because the table function has been eliminated, there is no need to perform the conversions that accompany the invocation of and return from the execution of the table function. At 1602 is shown a flowchart of the rewrite method. At 1625, the rewrite method is invoked by the compiler using the parameters from the table function. At 1627, the rewrite method determines whether the parameter values permit a rewrite. If they do not, the method does not return an SQL string (1629). If the parameter values do permit a rewrite, the rewrite method uses the parameters to write an SQL string that is equivalent to the table function (1631) and then returns the equivalent SQL string to the compiler. It is thus up to the writer of the rewrite method to determine when it is possible to write an SQL string that is equivalent to the table function.

How the SQL string is written in step 1631 of course depends on the parameters and the tables the query is written over. In the case of `RDF_MATCH`, generating the SQL string involves substantially

the same steps as generating the query when `RDF_MATCH` is executed. These steps are shown at 805 in FIG. 8. The difference between FIG. 8 and the processing shown in FIG. 16, of course, is that in FIG. 8, the SQL string for the query is generated at runtime and is executed *inside* the execution of `RDF_MATCH`; in FIG. 16, the SQL string for the query is generated at compile time and *replaces* the execution of `RDF_MATCH`. Because the SQL string replaces the execution of `RDF_MATCH`, the run time overhead resulting from the execution of the table function is eliminated. Included in the eliminated overhead are copying the results for the select list items of the containing SQL query to the respective attributes of the table function return object instance, passing the resulting object instance via the table function infrastructure, and remapping the passed object instance to the selected list items.

Other examples of the use of ODCITableRewrite: FIG. 20

FIG. 20 contains other examples of the use of `ODCITableRewrite`. At 2001 is shown how `ODCITableRewrite` may be used to replace an invocation of the table function `tab_func` with an SQL string. The original query with the invocation of `tab_func` is shown at 2003; the query that is generated when `tab_func` is executed is shown at 2005; the original query with the `TABLE` construct and the invocation of `tab_func` replaced by query string 2005 is shown at 2007.

Table functions may be used with SQL constructs other than the `TABLE` construct. The `TABLE` construct and other such constructs will be termed in the following *table function containers*. The effect in the `TABLE` construct and elsewhere is to *parameterize* the table function container, i.e., what result rows are returned by the table function container is determined by the parameters used in the table function. At 2009 in FIG. 20 is shown how a parameterized view may be used as a table function container. The parameterized view is called `summaries`. It returns a summary for a given period of time from a given table, with the given table and given period of time being determined by the parameterized view's parameters. The SQL-DDL for creating the parameterized view is shown at 2011. `USING` clause 2013 specifies how the result rows specified by the view will be obtained; in this case they are obtained by executing the table function `sum_tab_function` 2012; the parameter `fact_table` specifies a table from which the summary is going to be made and the parameter `time_granularity` specifies the period of time. At 2015 is shown a `SELECT` statement that obtains its result rows from the parameterized

view summaries 2016. The parameters 2014 in summaries are the parameters required for sum_tab_function 2012. Here, the table is a table of sales and the time period is a year.

If the table function has a rewrite method, the summaries parameterized view may be replaced by an SQL string generated by the rewrite method in the same fashion that the TABLE construct is replaced in the first example. The SQL string generated by the rewrite method for sum_tab_function 2012 is shown at 2017; and 2019 is shown the SELECT statement of 2015 in which parameterized view 2016 has been replaced by string 2017.

Conclusion

The foregoing *Detailed Description* has disclosed to those skilled in the relevant technologies how a TABLE function in a relational database system may be used to integrate queries written in non-SQL languages into a relational database system and has further disclosed the best mode presently known to the inventors of using a TABLE function in this fashion. In the *Detailed Description*, the technique is used to integrate RDF models and queries on the models made using RDF patterns into a relational database system, but as will be immediately apparent to those skilled in the relevant arts, the techniques described herein can be used for queries written in other non-SQL languages. Implementations of the techniques will of course vary to take the nature of the kind of database the query is applied to and the nature of the query into account. The implementation in the *Detailed Description* is also intended for a specific relational database system and is determined in considerable degree by characteristics of that relational database system. Finally, as is generally the case with inventions implemented in software, the implementer has wide latitude concerning the details of his or her implementation, including such details as the forms of tables, the form of the API, and the optimization techniques employed.

For all of the foregoing reasons, the *Detailed Description* is to be regarded as being in all respects exemplary and not restrictive, and the breadth of the invention disclosed here in is to be determined not from the *Detailed Description*, but rather from the claims as interpreted with the full breadth permitted by the patent laws.

What is claimed is:

1 1. Apparatus for integrating a non-SQL query written in a non-SQL language into a
2 relational database management system of the type wherein an SQL query may contain
3 an invocation of a table function and the table function, when executed, returns a set of
4 rows to the containing query,
5 the apparatus comprising:

6 a table function invocation in the SQL query which includes a parameter that
7 specifies the non-SQL query; and

8 query execution code associated with the table function that, when executed,
9 executes the non-SQL query on data that is accessible to the relational database
10 management system and returns results of the execution of the non-SQL query in the
11 returned set of rows.

1 2. The apparatus set forth in claim 1 further comprising:

2 SQL query string generating code associated with the table function that, when
3 executed, creates an SQL query string that is equivalent to the non-SQL query,
4 the relational database management system executing the query string generating code
5 and replacing the table function invocation in the containing query with the generated
6 SQL query string prior to executing the containing query.

1 3. A storage device that is accessible to a processor, the storage device being
2 characterized in that:

3 the storage device contains code which, when executed by the processor,
4 implements the apparatus set forth in claim 1.

1 4. Apparatus for integrating a non-SQL query written in a non-SQL language into a
2 relational database management system of the type wherein an SQL query may contain
3 an invocation of a table function and the table function, when executed, returns a set of
4 rows to the containing query,
5 the apparatus comprising:

6 a table function invocation in the SQL query which includes a parameter that
7 specifies the non-SQL query; and

8 SQL query string generating code associated with the table function that, when
9 executed, creates an SQL query string that is equivalent to the non-SQL query,
10 the relational database management system executing the query string generating code
11 and replacing the table function invocation in the containing query with the generated
12 SQL query string prior to execution of the containing query.

1 5. A storage device that is accessible to a processor, the storage device being
2 characterized in that:

3 the storage device contains code which, when executed by the processor,
4 implements the apparatus set forth in claim 4.

1 6. A method of integrating a non-SQL query written in a non-SQL language into a
2 relational database system of the type wherein an SQL query may contain an invocation
3 of a table function, the table function, when executed, returning a set of rows to the
4 containing SQL query and the method comprising the steps performed in the relational
5 database system of::

6 receiving a containing SQL query with a table function invocation which includes
7 a parameter that specifies the non-SQL query;

8 during execution of the containing SQL query, executing the specified non-SQL
9 query on data that is accessible to the table function; and

10 returning results of the execution of the non-SQL query in the set of returned
11 rows.

1 7. The method set forth in claim 6 further comprising the steps performed prior to
2 execution of the received containing SQL query of:

3 generating an SQL query string that is equivalent to the specified non-SQL:
4 query; and

5 replacing the invocation of the table function in the received containing SQL
6 query with the SQL query string.

1 **8.** A data storage device, characterized in that: the data storage device contains code
2 which when executed by a processor performs a method set forth in claim 6.

1 **9.** A method of integrating a non-SQL query written in a non-SQL language into a
2 relational database system of the type wherein an SQL query may contain an invocation
3 of a table function, the table function, when executed, returning a set of rows to the
4 containing SQL query and the method comprising the steps performed in the relational
5 database system of:

6 receiving a containing SQL query with a table function invocation which
7 includes a parameter that specifies the non-SQL query;

8 prior to execution of the containing SQL query, making an SQL query string that
9 returns results of the non-SQL query; and

10 replacing the table function invocation in the containing query with the SQL
11 query string.

1 **10.** A data storage device, characterized in that: the data storage device contains code
2 which when executed by a processor performs the method set forth in claim 9.

1 **11.** A method of integrating a query on a set of RDF triples into a relational database
2 system, the query being expressed as an RDF pattern, the relational database system
3 being of the type wherein a query may contain an invocation of a table function, the table
4 function, when executed, returning a set of rows to the containing query, and
5 the method comprising the steps performed in the relational database system of:

6 receiving a containing query that includes a table function invocation with a
7 parameter that specifies the RDF pattern;

8 during execution of the containing query, obtaining results that are equivalent to
9 those specified by the RDF pattern from RDF triples accessible to the database system
10 and returning the results in the returned rows.

1 **12.** The method set forth in claim 11 wherein:

2 the table function further has a parameter that specifies a RDF rule base of RDF
3 rules;
4 the RDF rule base is accessible to the relational database management system;
5 and
6 in the step of obtaining results, the results are further obtained from RDF triples
7 that are inferred using RDF rules from the specified RDF rule base.

1 **13.** The method set forth in claim 12 further comprising the step of:
2 inferring the inferred RDF triples and making them accessible to the relational
3 database management system prior to executing the containing query.

1 **14.** The method set forth in claim 12 wherein:
2 the table function's parameters further include a specification of an RDF model;
3 and
4 in the step of obtaining results, further obtaining results from inferred triples that
5 are inferred by applying RDF rules from the rule base to the RDF triples belonging to the
6 model.

1 **15.** The method set forth in claim 11 wherein:
2 in the step of obtaining results, the results are obtained by querying an RDF
3 triples table of representations of RDF triples in the relational database management
4 system

1 **16.** The method set forth in claim 15 wherein:
2 the relational database system includes one or more optimization objects for
3 optimizing queries on the RDF triples table; and
4 in the step of obtaining results, the relational database management system uses
5 the optimization objects to optimize the query on the RDF triples table.

1 **17.** The method set forth in claim 16 wherein:
2 the query optimization objects include an index on the RDF triples table.

1 **18.** The method set forth in claim 17 wherein:

2 the RDF triples table includes a subject column whose values represent RDF
3 subjects, a predicate column whose values represent RDF predicates, and an object
4 column whose values represent RDF objects; and
5 the index is on the predicate, subject, and object columns.

1 **19.** The method set forth in claim 16 wherein:

2 the queries include self-joins on rows from the RDF triples table; and
3 the query optimization objects include a materialized view of a self-join on the
4 RDF triples table.

1 **20.** The method set forth in claim 16 wherein:

2 the RDF triples table includes a subject column whose values represent RDF
3 subjects, a predicate column whose values represent RDF predicates, and an object
4 column whose values represent RDF objects; and
5 the query optimization objects include a subject-property matrix materialized join
6 view which has a subject column and one or more predicate columns, the values in the
7 predicate columns of a row of the join view being values of RDF objects which the RDF
8 triples in the RDF triples table relate to the value in the row's subject column.

1 **21.** The method set forth in claim 15 wherein:

2 the relational database management system includes an RDF rules table that
3 contains representations of RDF rules belonging to one or more rule bases; and
4 in the step of obtaining results, the query on the RDF Triples table is additionally
5 applied to inferred RDF triples that are inferred by applying rules represented in the RDF
6 rules table to the representations in the RDF triples table.

1 **22.** The method set forth in claim 21 wherein:

2 the table function's parameters further include a specification of at least one of the
3 rule bases; and

4 in the step of obtaining results, the applied rules belong to the rule base specified
5 in the rule base specification.

1 **23.** The method set forth in claim 21 further comprising the step of:

2 prior to the step of receiving a containing query, making an inferred triples table
3 containing triples inferred by applying the rules to the representations in the RDF triples
4 table; and

5 in the step of obtaining results, executing the query on the inferred triples from
6 the inferred triples table.

1 **24.** The method set forth in claim 11 further comprising the steps performed prior to
2 executing the containing query of:

3 generating a query string that specifies rows containing results that are equivalent
4 to those specified by the RDF pattern; and

5 replacing the invocation of the table function with the generated query string.

1 **25.** The method set forth in claim 11 wherein:

2 the step of obtaining the rows is performed by executing code associated with the
3 table function invocation that obtains the results from the RDF triples that are accessible
4 to the relational database system.

1 **26.** A data storage device, characterized in that:

2 the data storage device contains code which, when executed by a processor,
3 performs the method set forth in claim 11.

1 **27.** Apparatus for integrating a query expressed as an RDF pattern on a set of RDF
2 triples into a relational database management system of the type wherein a query may
3 contain an invocation of a table function, the table function, when executed, returning a
4 set of rows to the containing query,

5 the apparatus comprising:

6 an RDF table function in the relational database system whose parameters include
7 a specification of the RDF pattern; and

8 code associated with the RDF table function that causes rows containing results
9 that are equivalent to those specified by the RDF pattern to be obtained from RDF triples
10 accessible to the database system and returned to the containing query.

1 **28.** The apparatus set forth in claim 27 wherein:

2 the table function's parameters further include a specification of a RDF rule base
3 of RDF rules;

4 the rule base is accessible to the table function; and

5 the RDF triples include triples that are inferred using RDF rules from the rule
6 base.

1 **29.** The apparatus set forth in claim 28 wherein:

2 the inferred RDF triples are inferred prior to execution of the RDF table function.

1 **30.** The apparatus set forth in claim 28 wherein:

2 the table function's parameters further include a specification of an RDF model;
3 and

4 the RDF triples include RDF triples belonging to the model and inferred triples
5 that are inferred by applying RDF rule from the rule base to the RDF triples belonging to
6 the model.

1 **31.** The apparatus set forth in claim 27 further comprising:

2 a RDF triples table in the relational database management system that contains
3 representations of the RDF triples,

4 the code causes the results to be obtained from the RDF triples table.

1 **32.** The apparatus set forth in claim 31 further comprising;

2 one or more optimization objects for optimizing the query on the representations
3 of the RDF triples in the RDF triples table.

1 **33.** The apparatus set forth in claim 32 wherein:

2 the query optimization objects include an index on the RDF triples table.

1 **34.** The apparatus set forth in claim 33 wherein:

2 the RDF triples table includes a subject column whose values represent RDF
3 subjects, a predicate column whose values represent RDF predicates, and an object
4 column whose values represent RDF objects; and

5 the index is on the predicate, subject, and object columns.

1 **35.** The apparatus set forth in claim 32 wherein:

2 the queries include self-joins on rows from the RDF triples table; and

3 the query optimization objects include a materialized view of a self-join on the
4 RDF triples table.

1 **36.** The apparatus set forth in claim 32 wherein:

2 the RDF triples table includes a subject column whose values represent RDF
3 subjects, a predicate column whose values represent RDF predicates, and an object
4 column whose values represent RDF objects; and

5 the query optimization objects include a subject-property matrix materialized join
6 view which has a subject column and one or more predicate columns, the values in the
7 predicate columns of a row of the join view being values of RDF objects which the RDF
8 triples in the RDF triples table relate to the value in the row's subject column.

1 **37.** The apparatus set forth in claim 31 further comprising:

2 an RDF rules table in the relational database management system that contains
3 representations of RDF rules belonging to one or more rule bases,
4 the code additionally causing the results to be obtained from inferred RDF triples that are
5 inferred by applying rules represented in the RDF rules table to the representations in the
6 RDF triples table.

1 **38.** The apparatus set forth in claim 37 wherein:

2 the table function's parameters further include a specification of at least one of the
3 rule bases; and

4 the applied rules belong to the rule base specified in the rule base specification.

1 **39.** The apparatus set forth in claim 37 further comprising:

2 an inferred triples table containing triples inferred by applying the rules to the
3 representations in the RDF triples table,

4 the code further causing results to be obtained from the inferred triples table.

1 **40.** The apparatus set forth in claim 27 wherein:

2 the code is first code associated with the RDF table function that, when executed,
3 creates a query string that specifies results equivalent to the results specified by the RDF
4 pattern,

5 the relational database management system executing the first code prior to execution of
6 the containing query and replacing the RDF table function invocation in the containing
7 query with the generated query string.

1 **41.** The apparatus set forth in claim 27 wherein:

2 the code is second code associated with the RDF table function that, when
3 executed, returns the results specified by the RDF pattern,

4 the relational database management system executing the second code when the
5 containing query is executed.

1 **42.** A storage device that is accessible to a processor, the storage device being
2 characterized in that:

3 the storage device contains code which, when executed by the processor,
4 implements the apparatus set forth in claim 27.

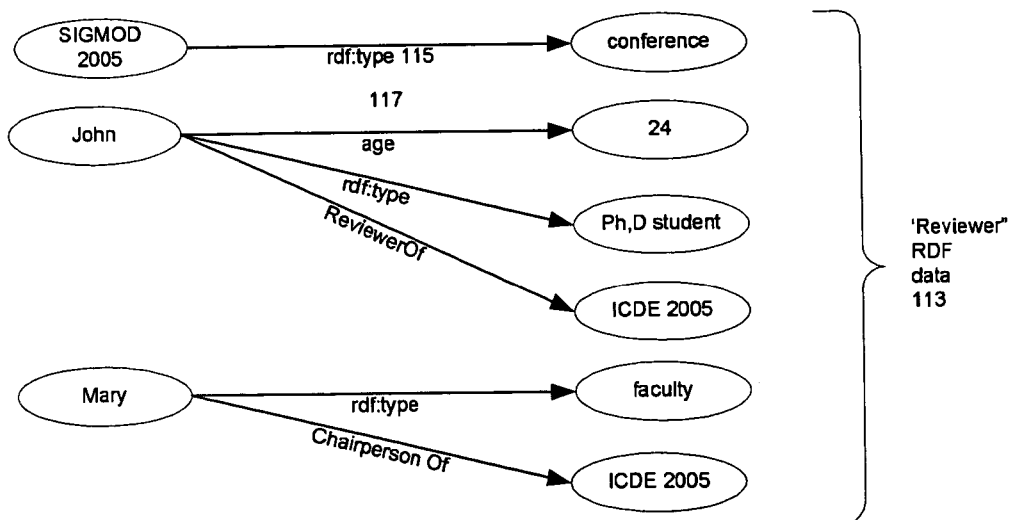
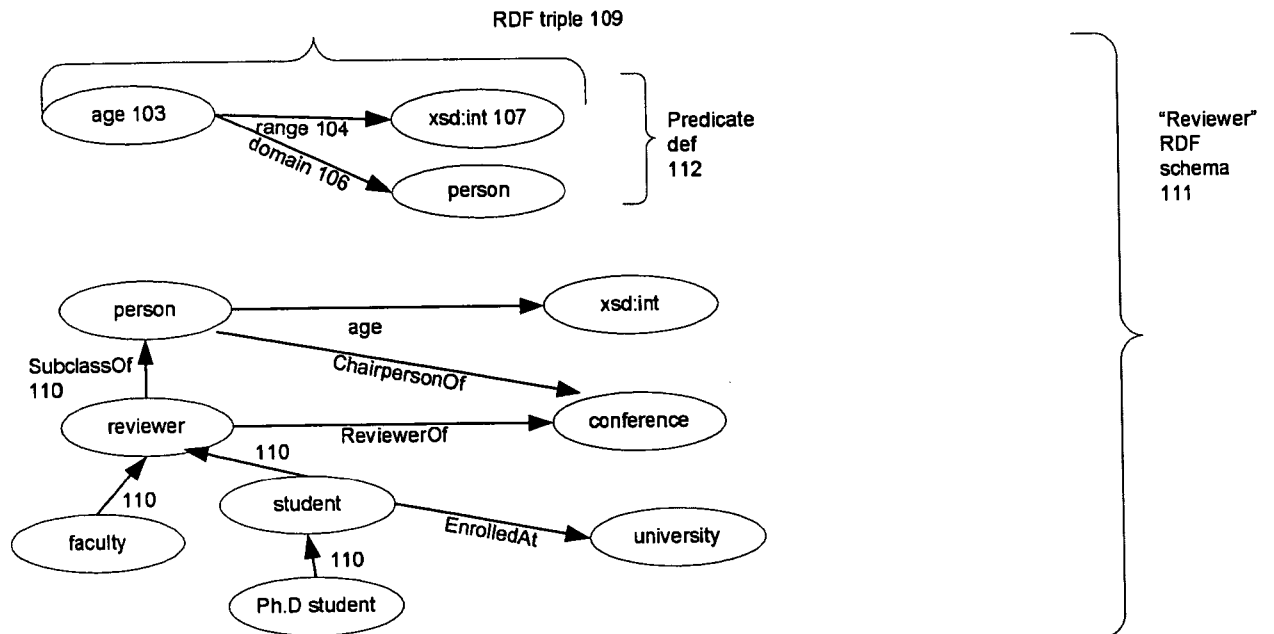
1 **43.** A materialized view that is an optimization for queries on a relational database table
2 wherein an entity is represented by a plurality of rows, each of the rows relating a type of
3 attribute and a value of the attribute to the entity,

4 the materialized view comprising:
5 a table wherein the entity is represented by a row, the row having a field for the
6 entity and a field for each type of attribute, with the field for a given type of attribute
7 containing the value of that attribute for the entity.

1 **44.** A storage device that is accessible to a processor, the storage device being
2 characterized in that:
3 the storage device contains code which, when executed by the processor,
4 implements the materialized view set forth in claim 43.

Abstract of the Disclosure

The TABLE function mechanism available in a RDBMS is used to integrate RDF models
5 into SQL queries. The table function invocation takes parameters including an RDF
pattern, an RDF model, and an RDF rule base and returns result rows to the SQL query
that contain RDF triples resulting from the application of the pattern to the triples of the
model and the triples inferred by applying the rule base to the model. The RDBMS
includes relational representations of the triples and the rules. Optimizations include
10 indexes and materialized views of the representations of the triples, precomputed inferred
triples, and a method associated with the TABLE function that rewrites the part of the
SQL query that contains the TABLE function invocation as an equivalent SQL string.
The latter technique is generally applicable to TABLE functions.



"Reviewer" RDF model 101

Fig. 1—Prior Art

Schema Triples		
Subject	Predicate	Object
Reviewer	rdfs:subClassOf	Person
Faculty	rdfs:subClassOf	Reviewer
Student	rdfs:subClassOf	Reviewer
Ph.D. Student	rdfs:subClassOf	Student
ChairpersonOf	rdfs:domain	Person
ChairpersonOf	rdfs:range	Conference
ReviewerOf	rdfs:domain	Reviewer
ReviewerOf	rdfs:range	Conference
Age	rdfs:domain	Person
Age	rdfs:range	xsd:int

201

Data Triples		
Subject	Predicate	Object
ICDE 2005	rdf:type	Conference
John	Age	24
John	rdf:type	Ph.D.student
John	ReviewerOf	ICDE 2005
Mary	rdf:type	Faculty
Mary	ChairpersonOf	ICDE 2005
Mary	Age	29
Tom	Age	22
Tom	rdf:type	Ph.D.student
Tom	ReviewerOf	ICDE 2005
Gary	Age	23
Gary	rdf:type	Ph.D.student
Gary	ReviewerOf	VLDB 2005
Bob	Age	21
Bob	rdf:type	Ph.D.student
Bob	ReviewerOf	VLDB 2005

203

205 { 207(1) ' (?r ReviewerOf ?c)
207(2) (?r rdf:type Ph.D.Student)
207(3) (?r Age ?a)',

Fig. 2 Prior Art



```
303 ('rb', -- rulebase name
305 'ChairpersonRule', -- rule name
307 '(?r ChairPersonOf ?c)', -- LHS pattern
308 NULL, -- filter condition
309 NULL, -- aliases
311 '(?r ReviewerOf ?c)') -- RHS pattern
```

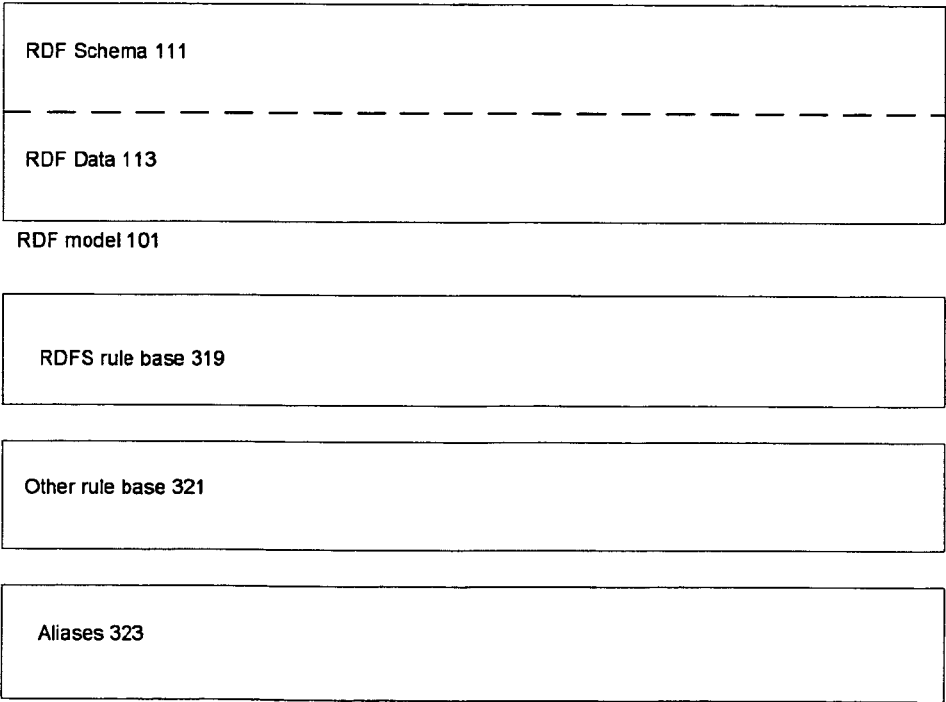
}

310

RDF rule 301

'(?r ReviewerOf ?c)'

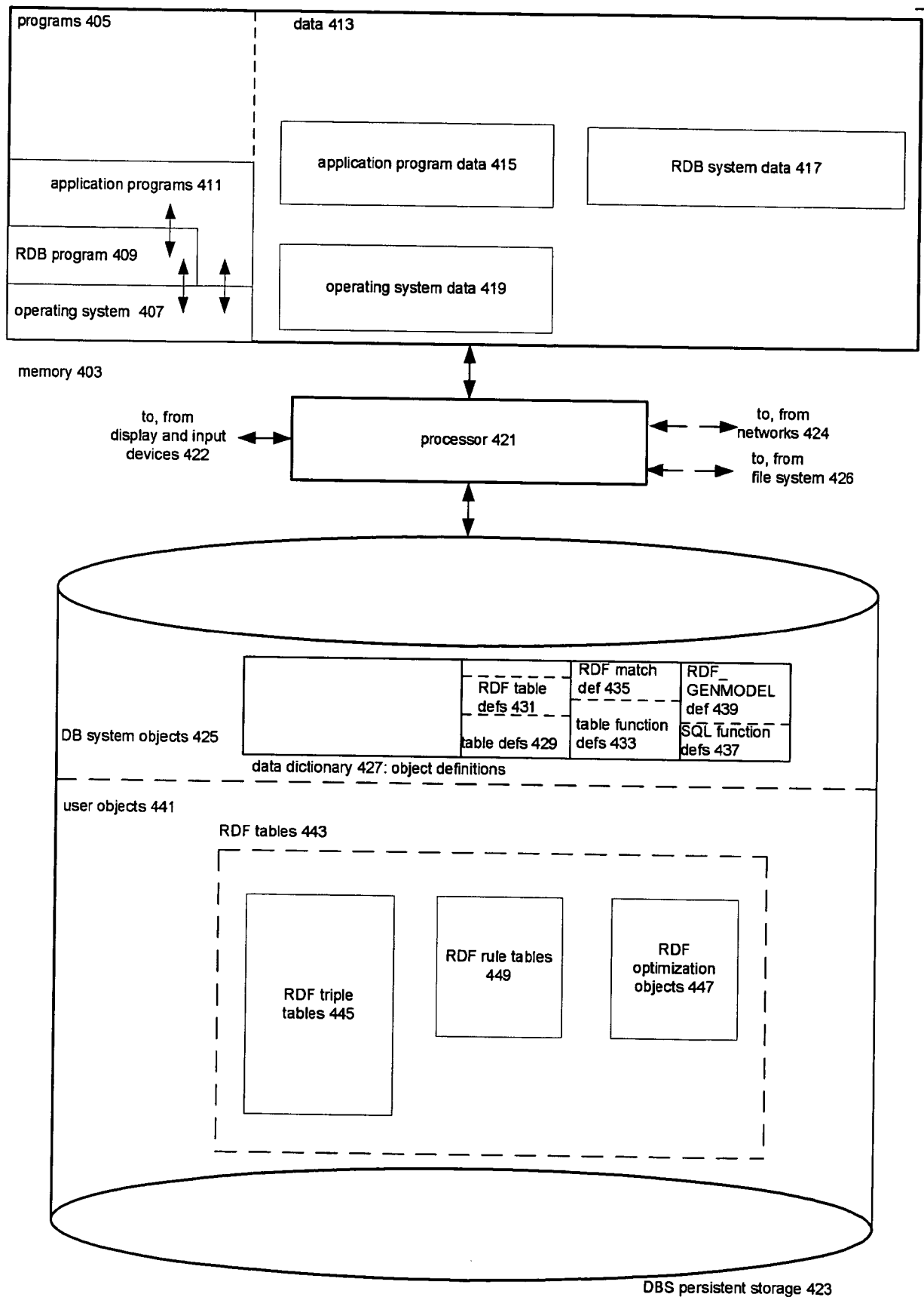
RDF pattern 312 that requires rule 301



RDF data set 313

Fig. 3 Prior Art





5/20

503 { 505 { 507 {
 SELECT t.r reviewer, t.c conf, t.a age
 FROM TABLE(RDF_MATCH(
 '(?r rdf:type Student)
 (?r ReviewerOf ?c)
 (?r Age ?a)',
 509 RDFModels ('reviewers'),
 511 513 NULL, NULL)) t 512
 WHERE t.a < 25;

RDF query 501

Student reviewer (r)	Conf (c)	c\$Type	Age(a)	A\$Type
John	ICDE 2005	URI	24	xsd:int
Tom	ICDE 2005	URI	22	xsd:int
Gary	VLDB 2005	URI	23	xsd:int
Bob	VLDB 2005	URI	21	xsd:int

rows 516 returned by RDF_Match 505

Student reviewer (r)	Conf (c)	Age(a)
John	ICDE 2005	24
Tom	ICDE 2005	22
Gary	VLDB 2005	23
Bob	VLDB 2005	21

result rows 514 returned by SELECT statement 503

517 {
 518 SELECT t.c conf,
 COUNT(*) row_count, AVG(t.a) avg_age
 505 FROM TABLE(RDF_MATCH(...)) t
 519 GROUP BY t.c
 521 ORDER BY avg_age;

RDF query with aggregation 515

Conf (c)	Count	Average
VLDB 2005	2	22
ICDE 2005	2	23

result rows 523 returned by SELECT statement 517

Fig. 5

6/20

	603	605	607	609
	ModelID	SubjectID	PropertyID	ObjectID
611	ReviewersID	ICDE2005ID	rdf:TypeID	ConferenceID
	ReviewersID	JohnID	AgeID	24ID
	ReviewersID	JohnID	rdf:TypeID	PhDStudentID
	ReviewersID	JohnID	ReviewerOfID	ICDE2005ID

IdTriples 601

24	<num. of row 621>
index entry 629	

idx_num 627

	615	617	623	625
	Internal ID	RDFVal	Canonical?	Literal Type
619	ReviewersID	"Reviewers"	y	String
620	ICDE2005ID	<ICDE2005 URI>	y	URI
	JohnID	<John URI>	y	URI
	AgeID	<Age URI>	y	URI
621	24ID	24	y	int

UriMap 613

RDF triple tables 445

Fig. 6

7/20

```
711  CreateRulesIndex (          /* Infer & store triples for
                                   specified models & rules */
                                   IndexName  VARCHAR2(30)
                                   Models      RDFModels,
                                   Rulebase    RDFRuleBases);

713  DropRulesIndex (          /* Remove inferred triples for
                                   specified models & rules */
                                   IndexName  VARCHAR2(30));
```

rulebase indexes 1009

Fig.7

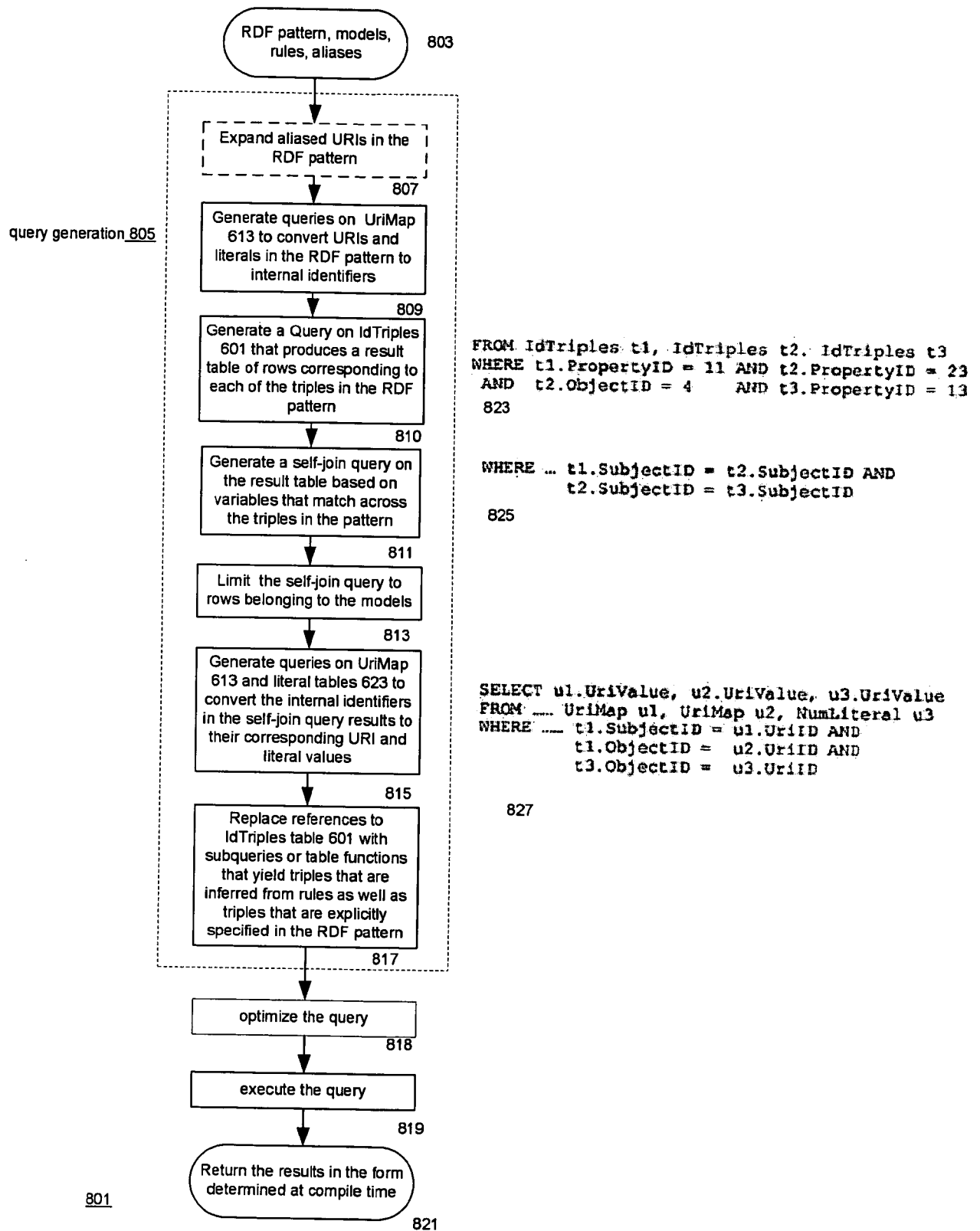


Fig. 8

9/20



	Data Triples		
	Subject	Predicate	Object
1	ICDE 2005	rdf:type	Conference
2	John	Age	24
3	John	rdf:type	Ph.D.student
4	John	ReviewerOF	ICDE 2005
5	Mary	rdf:type	Faculty
6	Mary	ChairpersonOf	ICDE 2005
7	Mary	Age	29
8	Tom	Age	22
9	Tom	rdf:type	Ph.D.student
10	Tom	ReviewerOf	ICDE 2005
11	Gary	Age	23
12	Gary	rdf:type	Ph.D.student
13	Gary	ReviewerOf	VLDB 2005
14	Bob	Age	21
15	Bob	rdf:type	Ph.D.student
16	Bob	ReviewerOf	VLDB 2005

203

Subject-subject join 903: Row pairs: (1,1) (2,2) (2,3) (2,4) (3,2) (3,3) (3,4) (4,2) (4,3)(4,4) etc.

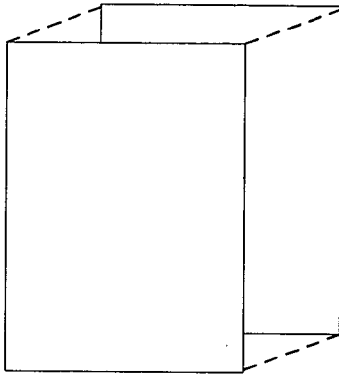
Subject-predicate join 905: No row pairs

Subject-object join 907: Row pairs: (1,4) (1,6)(1,10)

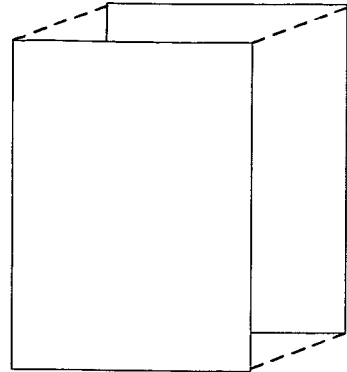
two-way joins 901

Fig. 9

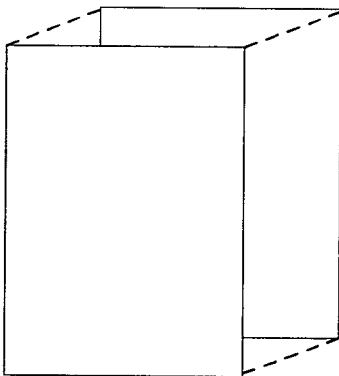




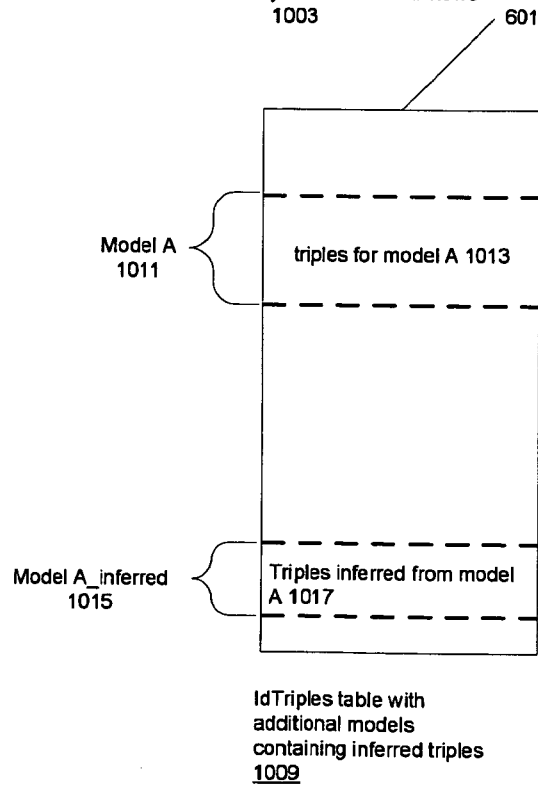
Generic join materialized views 1001



Subject-property matrix join materialized views 1003



IdTriples indexes 1005



11/20

Subject	Property	Object
John	rdf:type	Student
John	EnrolledAt	Univ1
John	Age	24
Pam	rdf:type	Student
Pam	EnrolledAt	Univ2
Pam	Age	22
Univ1	City	New York
Univ2	City	Los Angeles

RDF triples 1103

Subject	StudiesAt	Age
John	New York	24
Pam	Los Angeles	22

Ph.D student subject-property matrix join materialized
view for an Age predicate and an inferred Studies At
predicate 1105

```
'(?r rdf:type Student)
(?r enrolledAt ?u)
(?r age ?a)
(?u city ?city)'
```

RDF pattern 1107

1101

Fig. 11

12/20



Data Triples			
	Subject	Predicate	Object
1	ICDE 2005	rdf:type	Conference
2	John	Age	24
3	John	rdf:type	Ph.D.student
4	John	ReviewerOf	ICDE 2005
5	Mary	rdf:type	Faculty
6	Mary	ChairpersonOf	ICDE 2005
7	Mary	Age	29
8	Tom	Age	22
9	Tom	rdf:type	Ph.D.student
10	Tom	ReviewerOf	ICDE 2005
11	Gary	Age	23
12	Gary	rdf:type	Ph.D.student
13	Gary	ReviewerOf	VLDB 2005
14	Bob	Age	21
15	Bob	rdf:type	Ph.D.student
16	Bob	ReviewerOf	VLDB 2005

Rownums
1203

203

	1207	1209	1211	1213
1215	Predicate	Subject	Object	Rownum
	Age	John	24	2
	Age	Mary	29	7
	Age	Tom	22	8
	Age	Gary	23	11
	Age	Bob	21	14
	ChairpersonOf	Mary	ICDE 2005	6
	rdf:type	John	Ph.D. Student	3
	rdf:type	Mary	Faculty	5
	rdf:type	Tom	Ph.D. Student	9
	rdf:type	Gary	Ph.D. Student	12
	rdf:type	Bob	Student	15
	rdf:type	ICDE 2005	Conference	1
	ReviewerOf	John	ICDE 2005	4
	ReviewerOf	Tom	ICDE 2005	10
	ReviewerOf	Gary	VLDB 2005	13
	ReviewerOf	Bob	VLDB 2005	16

Predicate-subject-object
index 1205

1201

Fig. 12





`(?r ReviewerOf ?c)
 (?r Age ?a)'

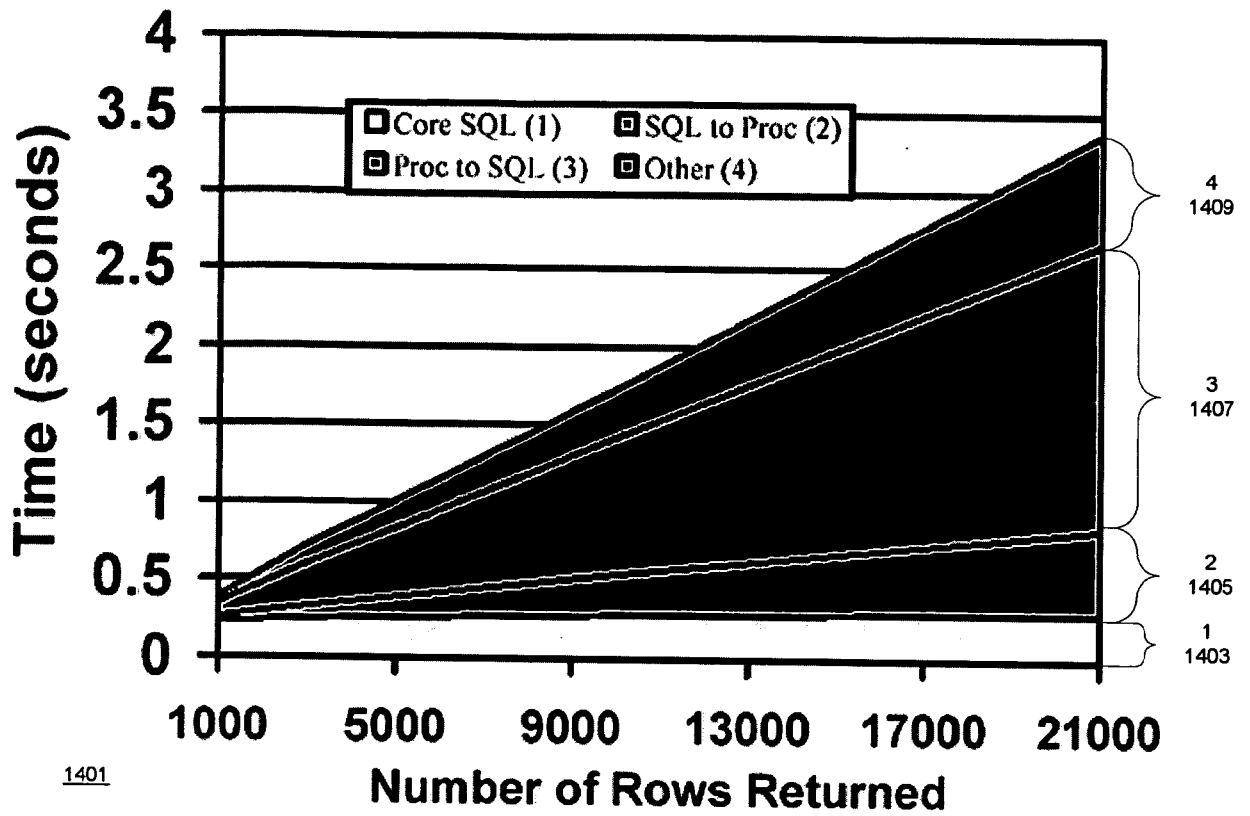
1303

`(?r ReviewerOf ?c)
 (?c rdf:type Conference)'

1305

1301

14/20



1415 { 1503 {
 SELECT t.a age
 FROM TABLE RDF_MATCH(
 '(?r, age ?a)'
 RDFModels('reviewers'),
 NULL,
 NULL) t
 WHERE t.a < 25

1419 { 1507 {
 SELECT t.a age
 FROM (SELECT ul.UriValue a
 FROM IdTriples t1, UriMap ul
 WHERE t1.PropertyID = 29 AND
 ul.UriID = ti.SubjectID) t
 WHERE t.a < 25

Fig. 14

1501 { 1503 { 1505 {
 SELECT t.a age
 FROM TABLE(RDF_MATCH(
 '(?r age ?a)'
 RDFModels('reviewers'),
 NULL,
 NULL)) t
 WHERE t.a < 25

1507 {
 SELECT ul.UriValue a
 FROM IdTriples t1, UriMap ul
 WHERE t1.PropertyID = 29 AND
 ul.UriID = t1.SubjectID

1509 { 1507 {
 SELECT t.a age
 FROM (SELECT ul.UriValue a
 FROM IdTriples t1, UriMap ul
 WHERE t1.PropertyID = 29 AND
 ul.UriID = t1.SubjectID) t
 WHERE t.a < 25

Fig. 15

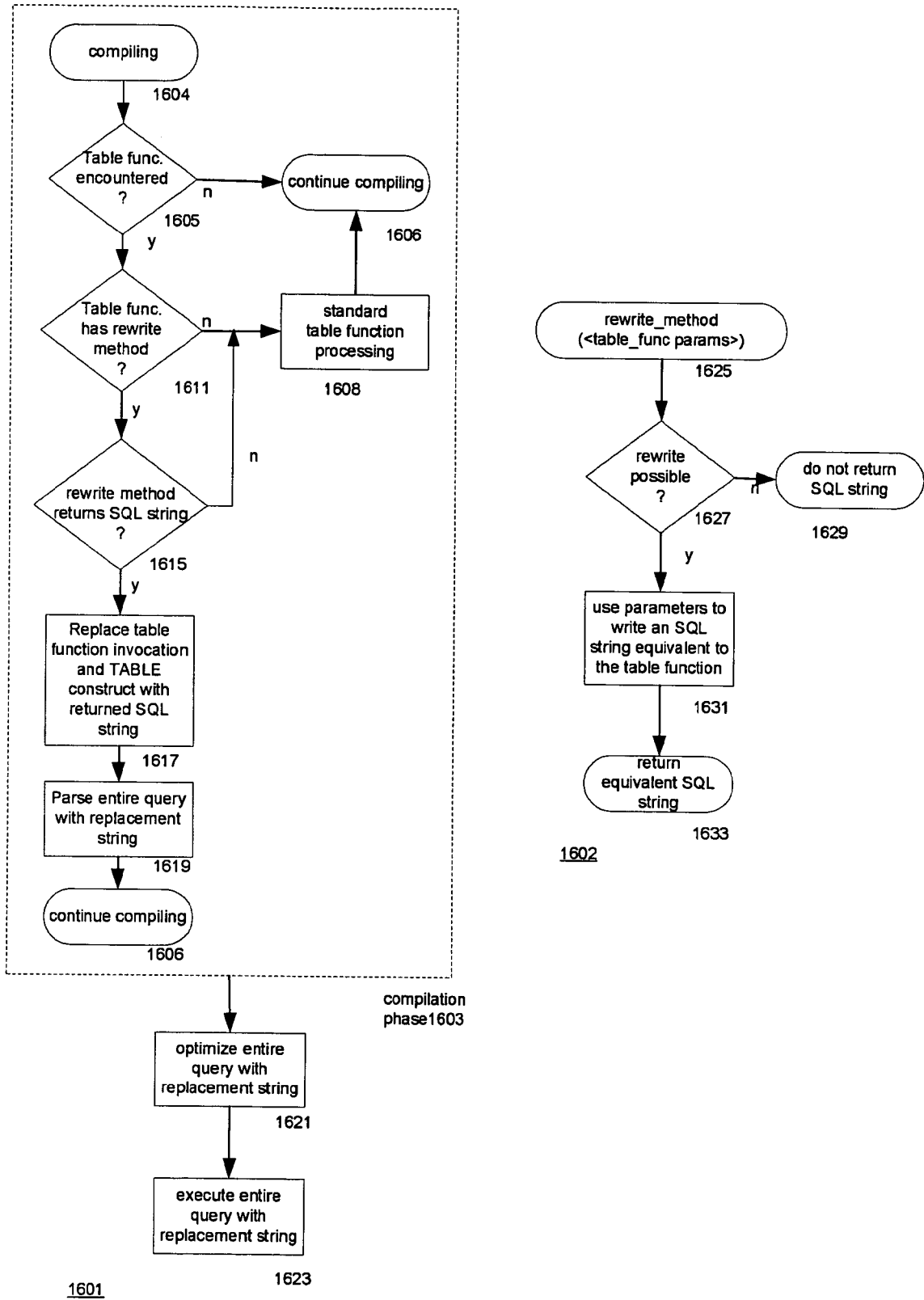


Fig. 16

1703		1705	
1707	RulebaseName	RuleName	
	<unique rule base name>	<unique rule name>	
	

rulebase table 1701

1711		1713	1715	1717
1719	RuleName	LeftPattern	Filter	RightPattern
	<unique rule name>	<pattern string>	<filter string>	<pattern string>

rule table 1709

```
1803      CreateRulebase (      /* Create a Rulebase */
                          RulebaseName  VARCHAR2 (30));

1805      DropRulebase   (      /* Delete a Rulebase */
                          RulebaseName  VARCHAR2 (30));

1807      InsertRuleIntoRulebase  (/* Add a rule to a RB */
                          RuleName      VARCHAR2 (30),
                          RulebaseName  VARCHAR2 (30));

1809      DeleteRuleFromRulebase  (/* Drop a rule from a RB */
                          RuleName      VARCHAR2 (30),
                          RuleBaseName  VARCHAR2 (30));

1811      CreateRule      (      /* Create a Rule */
                          Rulename      VARCHAR2 (30),
                          LeftPattern    RDFPattern,
                          Filter          CLOB,
                          RightPattern   RDFPattern
                          Aliases        RDFAliases);

1813      DropRule      (      /* Delete a Rule */
                          RuleName      VARCHAR2 (30));
```

Rulebase API 1801

Fig. 18

```

1903  RDFMviewCardinalities (
        Models      RDFModels,
        Rulebases   RDFRuleBases);

1905  RDFMviewCreate (      /* Create a materialized join view based on JoinColumns */
        MviewName   VARCHAR2(4000),
        Models      RDFModels,
        Rulebases   RDFRuleBases,
        JoinColumns VARCHAR2(2)); /* Values: 'SS', 'SO', 'SP', 'PP', 'OO' */

1907  RDFMviewDrop ( /* Delete a materialized view */
        MviewName   VARCHAR2(4000));

1909  RDFMatrixCreateMView ( /* Create a subject-property Matrix Mview */
        MviewName   VARCHAR2(4000),
        Models      RDFModels,
        Rulebases   RDFRuleBases,
        Pattern     RDFPattern,
        Filter       CLOB);

```

20/20

2003 { SELECT * FROM TABLE (tab_func('emp', 'dept' , 'dept_id')) t
WHERE t.emp_name LIKE 'p%';

2005 { SELECT e.name emp_name, d.dname dept_name
FROM employee e, department d
WHERE e.dept_id = d.dept_id;

2007 { SELECT * FROM ({ SELECT e.name emp_name, d.dname dept_name
2005 FROM employee e, department d
WHERE e.dept_id = d.dept_id) t
WHERE t.emp_name LIKE 'p%';

2001

2011 { CREATE VIEW summaries
USING sum_tab_function(fact_table VARCHAR,
2012 time_granularity VARCHAR);

2015 { SELECT * FROM summaries('sales', 'year');
2013 2016
2014

2017 { SELECT SUM(amount) sum_amount FROM sales
GROUP BY year;

2019 { SELECT * FROM (SELECT SUM(amount) sum_amount FROM sales
GROUP BY year);

2009

2017

Fig. 20

Attorney's Docket No.: oracle01.036

Patent

DECLARATION FOR PATENT APPLICATION

As a below named inventor(s), I hereby declare that:

My residence, mailing address and citizenship are as stated below, next to my name.

I believe I am the original, first, and sole inventor (if only one name is listed below) or an original, first, and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled

" Integrating RDF data into a relational database system "

the specification of which

 X is attached hereto.
 was filed on _____ as
United States Application Number _____,
or PCT International Application Number _____,
and was amended on _____
(if applicable)

I hereby state that I have reviewed and understand the contents of the above-identified specification, including the claims(s), as amended by any amendment referred to above.

I acknowledge the duty to disclose all information known to me to be material to patentability as defined in Title 37, Code of Federal Regulations, Section 1.56 (copy attached).

I hereby claim foreign priority benefits under Title 35, United States Code, Section 119(a)-(d), on any foreign application(s) for patent or inventor's certificate listed below and have also identified below any foreign application for patent or inventor's certificate having a filing date before that of the application on which priority is claimed:

Prior Foreign Application(s)

Priority
Claimed

<u> </u> (Number)	<u> </u> (Country)	<u> </u> (Day/Month/Year Filed)	<u> Yes </u>	<u> No </u>
<u> </u> (Number)	<u> </u> (Country)	<u> </u> (Day/Month/Year Filed)	<u> Yes </u>	<u> No </u>
<u> </u> (Number)	<u> </u> (Country)	<u> </u> (Day/Month/Year Filed)	<u> Yes </u>	<u> No </u>

I hereby claim the benefit under Title 35, United States Code, Section 119(e) of any United States provisional application(s) listed below

<u> </u> (Application Number)	<u> </u> (Filing Date)
<u> </u> (Application Number)	<u> </u> (Filing Date)

I hereby claim benefit under Title 35, United States Code, Section 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code, Section 112, I acknowledge the duty to disclose all information known to me to be material to patentability as defined in Title 37, Code of Federal Regulations, Section 1.56 (copy attached) which became available between the filing date of the prior application and the national or PCT International filing date of this application:

(Application Number)	(Filing Date)	(Status - patented, pending, abandoned)
(Application Number)	(Filing Date)	(Status - patented, pending, abandoned)
(Application Number)	(Filing Date)	(Status - patented, pending, abandoned)

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Full Name of Sole/First Inventor (given name, family name) Eugene Inseok Chong
Inventor's Signature Eugene S. Chong Date 4/11/2005
Residence Concord, MA Citizenship US
(City, State) (Country)
Mailing Address 9 Dunbar Way, Concord, MA 01742

Full Name of Second Inventor (given name, family name) Souripriya Das
Inventor's Signature Souripriya Das Date 4/11/2005
Residence Nashua, NH Citizenship US
(City, State) (Country)
Mailing Address 18 Sky Country Drive, Nashua, NH 03062

Full Name of Third Inventor (given name, family name) George Eadon
Inventor's Signature George Eadon Date 4/11/2005
Residence Nashua, NH Citizenship US
(City, State) (Country)
Mailing Address 20 Chadwick Circle, Apt. J, Nashua, NH 03062

Full Name of Fourth Inventor (given name, family name) Jegannathan Srinivasan
Inventor's Signature Jegannathan Date April 15, 2005
Residence Nashua, NH Citizenship India
(City, State) (Country)
Mailing Address 1 Hampshire Drive, Apt. F, Nashua, NH 03083

Title 37, Code of Federal Regulations, Section 1.56
Duty to Disclose Information Material to Patentability

(a) A patent by its very nature is affected with a public interest. The public interest is best served, and the most effective patent examination occurs when, at the time an application is being examined, the Office is aware of and evaluates the teachings of all information material to patentability. Each individual associated with the filing and prosecution of a patent application has a duty of candor and good faith in dealing with the Office, which includes a duty to disclose to the Office all information known to that individual to be material to patentability as defined in this section. The duty to disclose information exists with respect to each pending claim until the claim is canceled or withdrawn from consideration, or the application becomes abandoned. Information material to the patentability of a claim that is canceled or withdrawn from consideration need not be submitted if the information is not material to the patentability of any claim remaining under consideration in the application. There is no duty to submit information which is not material to the patentability of any existing claim. The duty to disclose all information known to be material to patentability is deemed to be satisfied if all information known to be material to patentability of any claim issued in a patent was cited by the Office or submitted to the Office in the manner prescribed by §§ 1.97(b)-(d) and 1.98. However, no patent will be granted on an application in connection with which fraud on the Office was practiced or attempted or the duty of disclosure was violated through bad faith or intentional misconduct. The Office encourages applicants to carefully examine:

- (1) Prior art cited in search reports of a foreign patent office in a counterpart application, and
- (2) The closest information over which individuals associated with the filing or prosecution of a patent application believe any pending claim patentably defines, to make sure that any material information contained therein is disclosed to the Office.

(b) Under this section, information is material to patentability when it is not cumulative to information already of record or being made of record in the application, and

- (1) It establishes, by itself or in combination with other information, a prima facie case of unpatentability of a claim; or
- (2) It refutes, or is inconsistent with, a position the applicant takes in:
 - (i) Opposing an argument of unpatentability relied on by the Office, or
 - (ii) Asserting an argument of patentability.

A prima facie case of unpatentability is established when the information compels a conclusion that a claim is unpatentable under the preponderance of evidence, burden-of-proof standard, giving each term in the claim its broadest reasonable construction consistent with the specification, and before any consideration is given to evidence which may be submitted in an attempt to establish a contrary conclusion of patentability.

(c) Individuals associated with the filing or prosecution of a patent application within the meaning of this section are:

- (1) Each inventor named in the application;
- (2) Each attorney or agent who prepares or prosecutes the application; and
- (3) Every other person who is substantively involved in the preparation or prosecution of the application and who is associated with the inventor, with the assignee or with anyone to whom there is an obligation to assign the application.

(d) Individuals other than the attorney, agent or inventor may comply with this section by disclosing information to the attorney, agent, or inventor.

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:

CHONG, et al.

Serial No.: Not Yet Assigned

Filed on: Herewith

Group Art Unit No.:

Not Yet Assigned

Examiner:

Not Yet Assigned

For: Integrating RDF data into a relational database system

POWER OF ATTORNEY
AND REVOCATION OF PREVIOUS POWERS

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

Oracle International Corporation, a California corporation having a place of business at 500 Oracle Parkway, M/S 50P7, Redwood Shores, California 94065, certifies that to the best of its knowledge and belief it is the assignee or is entitled to ownership of the entire right, title, and interest in and to the above-referenced patent application by virtue of an Assignment filed concurrently herewith and represents that the undersigned is a representative authorized and empowered to sign on behalf of Oracle Corporation, which hereby revokes all powers of attorney previously given and appoints the following attorney(s) and/or agent(s):

all practitioners associated with Customer No. 25247

with full power of substitution and revocation, to prosecute this application and to transact all business in the Patent and Trademark Office connected therewith. Send all future correspondence to the address associated with Customer No. 25247.

Assignee of Interest:

Oracle International Corporation

Dated: 15 April 2005

By:

Name:

Title:

Address of Assignee of Interest:

Oracle International Corporation
500 Oracle Parkway - M/S 50P7
Redwood Shores, CA 9406