Office de la Propriété Intellectuelle du Canada
Un organisme d'Industrie Canada

Canadian Intellectual Property Office
An agency of Industry Canada

CA 2468403 A1 2004/11/27

(21) 2 468 403

(12) DEMANDE DE BREVET CANADIEN
CANADIAN PATENT APPLICATION

(13) A1

(22) Date de dépôt/Filing Date: 2004/05/27

(41) Mise à la disp. pub./Open to Public Insp.: 2004/11/27

(30) Priorité/Priority: 2003/05/27 (2,429,910) CA

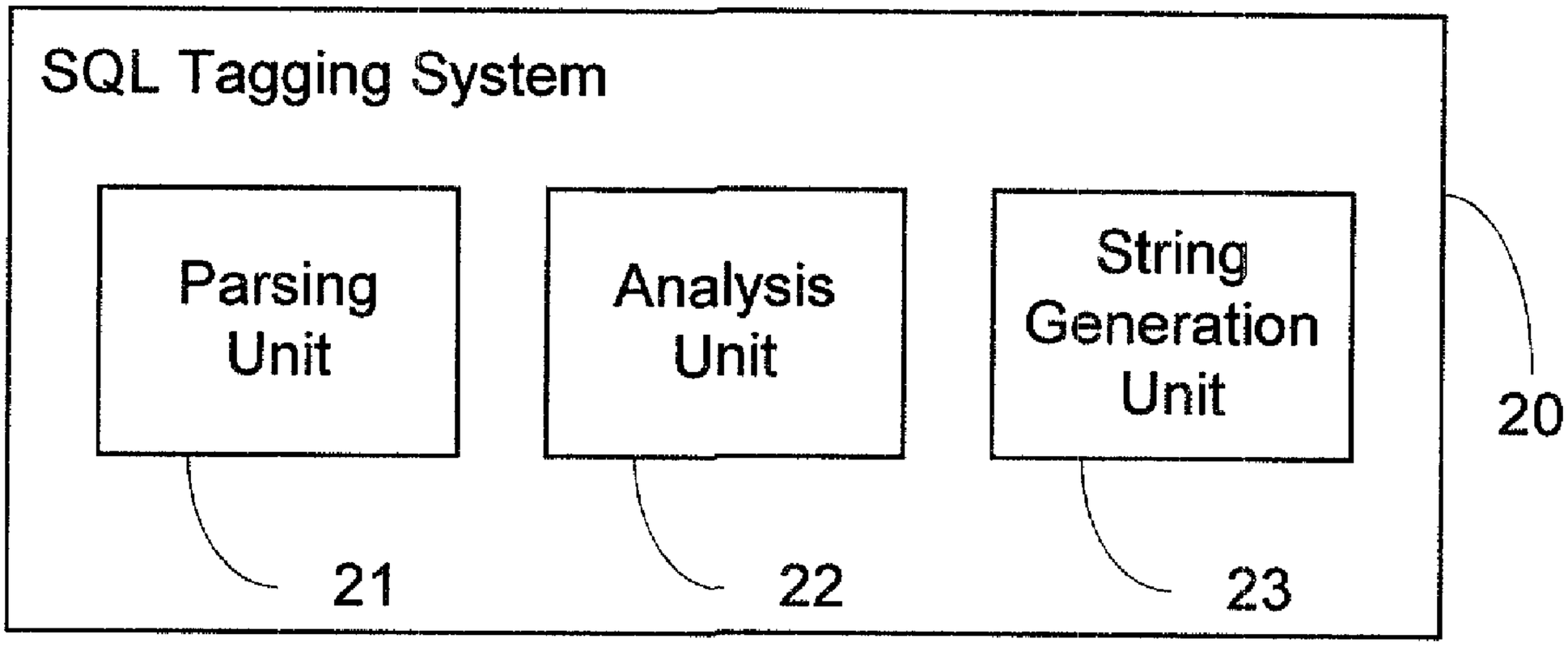
(51) Cl.Int.^{7}/Int.Cl.^{7} G06F 17/30

(71) Demandeur/Applicant:
COGNOS INCORPORATED, CA

(72) Inventeurs/Inventors:
STYLES, MICHAEL E., CA;
CAZEMIER, HENK, CA

(74) Agent: GOWLING LAFLEUR HENDERSON LLP

(54) Titre : SYSTEME ET METHODE DE TRANSFORMATION D'INTERROGATIONS
(54) Title: SYSTEM AND METHOD OF QUERY TRANSFORMATION



(57) Abrégé/Abstract:
A system for structured query language tagging is provided. The system comprises a parsing unit for parsing a structured query language string into components, an analysis unit for analyzing the components and applying associated tags to the components, and a string generation unit for concatenating the components with associated tags into a new string.

Abstract

A system for structured query language tagging is provided. The system comprises a parsing unit for parsing a structured query language string into components, an analysis unit for analyzing the components and applying associated tags to the components, and a string generation unit for concatenating the components with associated tags into a new string.

System and Method of Query Transformation

FIELD OF THE INVENTION

The invention relates generally to data access middleware, and in particular to a system and method of query transformation.

BACKGROUND OF THE INVENTION

A typical data access environment has a multi-tier architecture. For description purposes, it can be separated into three distinct tiers:

- Web server
- Applications
- Data

The tiers are based on business function, and are typically separated by firewalls. Client software, such as a browser or a report-authoring tool, sits above the tiers.

The web server contains a firewall and one or more gateways. All web communication is performed through a gateway. A gateway is responsible for passing on requests to the application server, in tier 2, for execution.

The applications tier contains one or more application servers. The application server runs requests, such as reports and queries that are forwarded by a gateway running on the web server. Typically, one of the components of the applications tier is a query engine, which is data access middleware that provides universal data access to a variety of heterogeneous database systems. The query engine formulates queries (typically SQL) and passes them on to the data tier, through a native database API (such as ODBC) for execution.

The data tier contains database management systems (DBMS), which manage raw data stored in a database. Examples of such systems include Oracle, DB2, and Microsoft SQL Server.

Although a multi-tier architecture can be configured in several different ways, a typical configuration places each tier on a separate computer (server). A database server is typically a "high end" server, and thus can process queries at a relatively fast speed. An application server cannot generally process queries as quickly as a database server.

In order to solve many business questions, a query engine may generate SQL queries that utilize the SQL/OLAP technology introduced in the SQL-99 standard.

However, many database systems do not support this technology. Thus, the SQL queries would have to be performed on the report server that is generally slower than the database server. It is desirable to have as much processing performed on the database server.

There is a need to prevent or reduce the amount of local (application server) processing required to process a query. In the past, the application would be responsible for generating separate SQL queries involving the GROUP BY operator to compute aggregates over different partitions and stitching together the results. Quite often, this is quite difficult since it involves multiple queries and post processing.

One way of overcoming this problem is for the query engine to generate separate GROUP BY queries for aggregates computed over different partitions, generate a separate query to retrieve detail information, and then stitch together the results to produce the desired report. Unfortunately, this problem requires processing time on the report server. It is desirable to have a way of transferring the SQL queries to the database server with minimal processing on the report server.

It is desirable to produce a meaningful result from a many-to-one-to-many relationship using a single structured query language (SQL) statement. In the past, many-to-one-to-many query relationships are dealt with by issuing two separate queries and then the stitching the results together.

SUMMARY OF THE INVENTION

It is an object of the present invention to provide a method of query transformation in a database system that does not support SQL-99 standard.

In accordance with an embodiment of the present invention, there is provided a structured query language tagging system for tagging structured query language commands. The system comprises a parsing unit for parsing a structured query language string into components, an analysis unit for analyzing the components and applying associated tags to the components, and a string generation unit for concatenating the components with associated tags into a new string.

In accordance with another embodiment of the present invention, there is provided a method of tagging structured query language commands. The method comprises the steps of parsing an incoming structured query language statement and producing an input tree representation, and traversing the input tree representation while generating a structured query language string with embedded extensible markup language tags.

In accordance with an embodiment of the present invention, there is provided a method of tagging structured query language commands. The method comprises the steps of parsing a structured query language string into components, analyzing the components and applying associated tags to the components, and concatenating the components with associated tags into a new string.

In accordance with an embodiment of the present invention, there is provided a WITH clause transformation system comprising a WITH clause analysis module for analysing structured query language / online analytical programming (SQL/OLAP) windowed aggregates that are not supported by a target database system, and a WITH clause transformation module for transforming SQL/OLAP windowed aggregates into semantically equivalent standard aggregate functions that are supported by the target database system.

In accordance with an embodiment of the present invention, there is provided a method of WITH clause transformation. The method comprises the steps of analysing a query containing SQL/OLAP windowed aggregates that are not supported by a target database system, and transforming the query into a semantically equivalent query, having a WITH clause, that is supported by the target database system.

In accordance with an embodiment of the present invention, there is provided a parallel detail join system for merging elements in a plurality of tables. The system comprises a parallel detail analysis module for obtaining related elements in a plurality of tables, and a parallel detail transformation module for merging the related elements in a singe table. The related elements merged such that the number of rows of a single element equals a largest cardinality of the single element and the number of columns equals the number of tables.

In accordance with an embodiment of the present invention, there is provided a method of merging elements in a plurality of tables. The method comprises the steps of obtaining related elements in a plurality of tables, and merging the elements are merged into a single table such that the number of rows of a single element equals a largest cardinality of the single element and the number of columns equals the number of tables.

In accordance with an embodiment of the present invention, there is provided a method of merging elements in a plurality of tables. The method comprises the steps of analyzing elements in a plurality of tables, creating a single table, and placing the properties of each respective element in the single table in a number of rows equal to a

largest cardinality of the respective element. The single table having a number of columns equal to the number of tables plus one, and a number of rows equal to the total of the largest cardinality of the elements in each table.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 shows a typical data access environment.

Figure 2 shows an example of a structured query language tagging system, in accordance with an embodiment of the present invention.

Figure 3 shows in a flowchart an example of a method of tagging structured query language commands, in accordance with the structured query language tagging system.

Figure 4 shows in a flowchart another example of a method of tagging structured query language commands, in accordance with the structured query language tagging system.

Figure 5 shows a WITH clause query transformation system, in accordance with an embodiment of the present invention.

Figure 6 shows in a flowchart an example of a method of structured query language group transformation, in accordance with an embodiment of the group query transformation system.

Figure 7 shows an example of a parallel detail join system for merging elements in a plurality of tables, in accordance with an embodiment of the present invention.

Figure 8 shows in a flowchart an example of a method of merging elements in a plurality of tables, in accordance with an embodiment of the parallel detail join system.

Figure 9 shows in a flowchart another example of a method of merging elements in a plurality of tables, in accordance with an embodiment of the parallel detail join system.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Figure 1 shows a typical data access environment 10 for processing data. Typically, data is stored in a database 11. A DBMS running on a database server 12 accesses the raw data stored in the database 11. A query engine 15, running on an application server 12 is used to generate reports on the raw data and instruct the DBMS on the database server 12 to obtain information pertaining to the raw data in the database 11. The query engine 15 provides universal data access to a variety of heterogeneous

database systems. An end user uses a client application 14, running on a client workstation, to facilitate application server 13 operations.

In order to solve many business questions, a query engine 15 generates SQL queries that utilize the SQL/online analytical programming (OLAP) technology introduced in the SQL-99 standard. These SQL queries include SQL/OLAP functions (windowed aggregates). However, many database systems 12 do not support this technology. In order to prevent or reduce the amount of local (application server) processing required to process these types of queries, the query engine 15 attempts to generate semantically equivalent queries that can be processed on the database server 12 by the target database system. These semantically equivalent queries include standard aggregate functions and the GROUP BY operator.

Figure 2 shows an example of a structured query language (SQL) tagging system 20, in accordance with an embodiment of the present invention. The SQL tagging system 20 comprises a parsing unit 21 for parsing a structured query language string into components, an analysis unit 22 for analyzing the components and applying associated tags to the components, and a string generation unit 23 for concatenating the components with associated tags into a new string. Preferably, the parsing unit 21 comprises an extensible markup language (XML) parser built into a query engine application that implements the SQL tagging system 20. Alternatively, the SQL tagging system 20 may comprise a separately implemented SQL parser. The tagging of components may be performed by a universal data access (UDA). Components may be added to, or removed from, the SQL tagging system 20.

The SQL tagging system 20 may be implemented as a component of the report server 13, or as a component of the query engine 15 to a data access environment 10. Alternatively, the SQL tagging system 20 may be implemented as a component of the database server 12. A further alternative has the SQL tagging system 20 implemented as a separate component between the report server 13 and the database server 12 in a data access environment 10.

Figure 3 shows in a flowchart an example of a method of tagging SQL commands (30), in accordance with the structured query language tagging system 20. The method (30) begins with parsing an incoming SQL statement and producing an input tree (ITREE) representation (31). Next, the ITREE is traversed while generating a SQL string with the appropriate XML tags embedded (32). The method (30) is done (33).

Advantageously, the SQL tagging system 20 allows various components of a SQL statement to be extracted and analyzed by a query engine. The query engine can then us this information to simplify the SQL as much as possible, keeping only the minimal set of tables and joins needed to obtain values for the selected query items.

Furthermore, as described above, the query engine which implements the SQL tagging system 20 does not need to implement a full-blown SQL parser. The query engine can use its existing XML parser to construct a document object model (DOM) tree for the purpose of extraction and analysis of various components of a SQL statement.

Furthermore, this embodiment of the invention saves on development and maintenance costs since, as mentioned above, CQE does not need to develop (and therefore maintain) a SQL parser for producing a DOM tree. New functionality in the SQL language can be handled easily by simply introducing new XML tags.

Figure 4 shows in a flowchart another example of a method of tagging structured query language commands (40), in accordance with the structured query language system 20. The method (40) begins with parsing a SQL string into components (41). Once the components are parsed (41), the components may be analysed to apply associated tags to the components (42). Once the associated tags are applied to the components, the components with associated tags may be concatenated into a new string (43). The method (40) is done (44) and a query engine may use the new string. Steps may be added to, or removed from, the method (40).

In one embodiment of the SQL tagging system 20, SQL tagging involves parsing a SQL string and producing a new string containing XML tags. An example of a collection of tags is given below:

XML Tag	Description
<field>	Identifies a value expression in select list.
<column>	Identifies a column reference in value expression.
<alias>	Identifies a derived column name for value expression in select list or a correlation name appearing in the FROM clause.
<subquery>	Identifies a subquery.
<orderby>	Identifies a list of one or more sort columns in an ORDER BY clause

<groupby>	Identifies a list of one or more grouping columns in an GROUP BY clause
<having>	Identifies a HAVING clause predicate.
<summaryfilter>	Identifies a FILTER (summary filter) clause predicate.
<window>	Identifies a WINDOW clause.
<distinct>	Identifies the DISTINCT qualifier in a select list.
<qualify>	Identifies a QUALIFY predicate.
<filter>	Identifies a WHERE clause predicate.
<joinedtable>	Identifies a joined table (INNER JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN, or CROSS JOIN) in the FROM clause.
<derivedtable>	Identifies a derived table in the FROM clause.
<table>	Identifies a simple table reference in the FROM clause.
<nativesql>	Identifies database-specific SQL.
<passthroughSQL>	Identifies standalone database-specific SQL.
<procedure>	Identifies a stored procedure call.
<with>	Identifies a WITH clause specification.
<view>	Identifies a common table expression referenced in the FROM clause.

Example

Original SQL

```
SELECT  C1, SUM( C2 ) OVER ()
FROM (  SELECT  SNO C1, SUM( QTY ) OVER ( PARTITION BY SNO ) C2
        FROM SUPPLY ) T1
WHERE   C2 > 1000
```

Tagged SQL

```
SELECT  <field><column>C1</column></field>,
        <field>SUM( <column>C2</column> ) OVER ()</field>
FROM    <derivedtable> (
        SELECT <field><column>SNO</column><alias>C1</alias></field>,
            <field>
                SUM( <column>QTY</column> )
```

OVER (PARTITION BY **<column>**SNO**</column>**)

<alias>C2**</alias>**

</field>

FROM **<table>**SUPPLY**</table>**) **</derivedtable>** **<alias>**T1**</alias>**

WHERE **<filter><column>**C2**</column>** > 1000**</filter>**

Explanation

The above example shows the introduction of specialized XML tags (in bold) into the original SQL statement. Each tag identifies a particular part of the SQL statement. Advantageously, the tagged SQL can be easily parsed by any XML parser (the original SQL cannot) to produce a DOM tree.

WITH Clause Transformation

Figure 5 shows a WITH clause transformation system 50, in accordance with an embodiment of the present invention. The WITH clause query transformation system 50 comprises a WITH clause analysis module 51 for analysing SQL/OLAP windowed aggregates that are not supported by a target database system, and a WITH clause transformation module 52 for transforming SQL/OLAP windowed aggregates into semantically equivalent standard aggregate functions that are supported by the target database system.

The WITH clause transformation system 50 may be implemented as a sub-system of the query engine 15 in the data access environment 10. This transformation system 50 may generate queries that can be processed in their entirety on the database server 12, or queries that require processing on both the application server 13 and the database server 12.

Advantageously, the WITH clause transformation system 50 reduces processing that might otherwise be required on the application server, thereby improving performance in many cases. The WITH clause transformation system uses the WITH clause and common table expressions to express a semantically equivalent query.

The group query transformation of a SQL-99 query involves mapping windowed OLAP functions into a semantically equivalent query involving derived tables and the standard WITH clause.

Figure 6 shows in a flowchart an example of a method of WITH clause transformation (60), in accordance with an embodiment of the WITH clause transformation system 50. The method (60) begins with analysing a query containing SQL/OLAP windowed aggregates that are not supported by a target database system (61). Next, the query is transformed into a semantically equivalent query, having a WITH clause, that is supported by the target database system (62). The method (60) is done (63).

Example 1

Original Query

```
SELECT      SNO, PNO, SUM( QTY ) OVER ( PARTITION BY SNO ),
            SUM( QTY ) OVER ( PARTITION BY SNO, PNO )
FROM        SUPPLY
```

Transformed Query

```
WITH  T0 AS ( SELECT      SNO C0, PNO C1, SUM( QTY ) C2
                  FROM        SUPPLY
                  GROUP       BY SNO, PNO ),
              T1 AS ( SELECT      C0, SUM( C2 ) C1
                  FROM        T0
                  GROUP       BY C0 ),
              T2 AS ( SELECT      SNO C0, PNO C1
                  FROM        SUPPLY )
    SELECT    T2.C0, T2.C1, T0.C1, T1.C2
    FROM      T0, T1, T2
    WHERE     ( T2.C0 = T0.C0 OR ( T2.C0 IS NULL AND T0.C0 IS NULL ) )
    AND       ( T2.C0 = T1.C0 OR ( T2.C0 IS NULL AND T1.C0 IS NULL ) )
    AND       ( T2.C1 = T1.C1 OR ( T2.C1 IS NULL AND T1.C1 IS NULL ) )
```

Example 2

Original Query

```
SELECT      DISTINCT SNO, PNO, SUM( QTY ) OVER ( PARTITION BY SNO ),
            SUM( QTY ) OVER ( PARTITION BY SNO, PNO )
FROM        SUPPLY
```

Transformed Query

```
WITH T0 AS (        SELECT      SNO C0, PNO C1, SUM( QTY ) C2
                    FROM SUPPLY
                    GROUP       BY SNO, PNO ),
     T1 AS (        SELECT      C0, SUM( C2 ) C1
                    FROM T0
                    GROUP       BY C0 )
SELECT      T2.C0, T2.C1, T0.C1, T1.C2
FROM        T0, T1, T2
WHERE       ( T1.C0 = T0.C0 OR ( T1.C0 IS NULL AND T0.C0 IS NULL ) )
```

Example 3

Original Query

```
SELECT      SNO, PNO, QTY,
            SUM( QTY ) OVER (), SUM( QTY ) OVER ( PARTITION BY SNO ),
            SUM( QTY ) OVER ( PARTITION BY SNO, PNO ),
            RANK() OVER ( ORDER BY QTY DESC )
FROM        SUPPLY
```

Transformed Query

```
WITH        T0 AS (     SELECT      SNO C0, PNO C1, SUM( QTY ) C2
                        FROM        SUPPLY
                        GROUP       BY SNO, PNO ),
            T1 AS (     SELECT      SNO C0, SUM( C2 ) C1
                        FROM        T0
                        GROUP       BY SNO ),
            T2 AS (     SELECT      SUM( C1 ) C2
                        FROM        T1 ),
            T3 AS (     SELECT      SNO C0, PNO C1, QTY C2
                        FROM        SUPPLY )
SELECT      T0.C0, T0.C1, T0.C2, T2.C0, T1.C1, T0.C2, RANK() OVER ( ORDER
BY T3.C2 DESC )
FROM        T0, T1, T2, T3
WHERE       ( T3.C0 = T1.C0 OR ( T3.C0 IS NULL AND T1.C0 IS NULL ) )
```

```
AND     ( T3.C0 = T2.C0 OR ( T3.C0 IS NULL AND T2.C0 IS NULL ) )
AND     ( T3.C1 = T2.C1 OR ( T3.C1 IS NULL AND T2.C1 IS NUL L ) )
```

Parallel Detail Join

Parallel detail join solves the problem of producing a meaningful result from a many-to-one-to-many relationship using a single structured query language (SQL) statement.

Figure 7 shows an example of a parallel detail join system 70 for merging elements in a plurality of tables, in accordance with an embodiment of the present invention. The parallel detail join system 70 comprises a parallel detail analysis module 71 for obtaining related elements in a plurality of tables, and a parallel detail transformation module 72 for merging the related elements in a singe table. The related elements may have different cardinalities in each original table. The related elements are merged such that the number of rows equals the largest cardinality and the number of columns equals the number of tables.

The parallel detail join system 70 may be implemented as a component of the report server 13, or as a component of the query engine 15 to a data access environment 10. Alternatively, the parallel detail join system 70 may be implemented as a component of the database server 12. A further alternative has the parallel detail join system 70 implemented as a separate component between the report server 13 and the database server 12 in a data access environment 10.

Figure 8 shows in a flowchart an example of a method of merging elements in a plurality of tables (80), in accordance with an embodiment of the parallel detail join system 60. The method (80) begins with obtaining related elements in a plurality of tables (81). Next, the elements are merged into a single table (82). The method (80) is done (83).

Advantageously, the parallel detail join system reduces processing that might otherwise be required on the application server, thereby improving performance in many cases.

Figure 9 shows in a flowchart another example of a method of merging elements in a plurality of tables (90), in accordance with an embodiment of the parallel detail join system 60. The method (90) begins with analyzing elements in a plurality of tables (91).

Next, a single table is created, having a number of columns equal to the number of tables plus one, and a number of rows equal to the total of the largest cardinality of the elements in each table (92). Next, the properties of each respective element are placed in the table in a number of rows equal to the largest cardinality of the respective element (93). The method (90) is done (94).

Assume we have the following sample database. There is a one-to-many relationship between **EMPLOYEES** and **BILLINGS**, and a one-to-many relationship between **EMPLOYEES** and **SKILLS.** The **BILLINGS** and **SKILLS** tables may have different cardinalities.

EMPLOYEES

ID	NAME
1	Stan
2	Mike
3	John

BILLINGS

ID	AMOUNT
1	100
1	400
1	500
3	600

SKILLS

ID	SKILL
1	Cobol
1	C
2	Pascal
2	Visual Basic

The desired result is shown below:

ID	NAME	AMOUNT	SKILL
1	Stan	100	Cobol
1	Stan	400	C
1	Stan	500	
2	Mike		Pascal
2	Mike		Visual Basic
3	John	600	

This can be accomplished with the following SQL-99 query:

```
SELECT COALESCE( T1.ID, T2.ID ), COALESCE( T1.NAME, T2.NAME ), T1.AMOUNT,
                T2.SKILL
FROM ( SELECT T1.ID, T1.NAME, T2.AMOUNT,
                ROW_NUMBER() OVER ( PARTITION BY T1.ID ORDER BY T1.ID ) RS
        FROM    EMPLOYEES T1 LEFT OUTER JOIN BILLINGS T2 ON T1.ID = T2.ID )
T1
        FULL OUTER JOIN
     ( SELECT   T1.ID, T1.NAME, T2.SKILL,
                ROW_NUMBER() OVER ( PARTITION BY T1.ID ORDER BY T1.ID )
RS
        FROM    EMPLOYEES T1 LEFT OUTER JOIN SKILLS T2 ON T1.ID = T2.ID ) T2
        ON      T1.ID = T2.ID
        AND     T1.RS = T2.RS
```

The systems and methods according to the present invention may be implemented by any hardware, software or a combination of hardware and software having the above described functions. The software code, either in its entirety or a part thereof, may be

stored in a computer readable memory. Further, a computer data signal representing the software code that may be embedded in a carrier wave may be transmitted via a communication network. Such a computer readable memory and a computer data signal are also within the scope of the present invention, as well as the hardware, software and the combination thereof.

While particular embodiments of the present invention have been shown and described, changes and modifications may be made to such embodiments without departing from the true scope of the invention.

WHAT IS CLAIMED IS:

1. A structured query language tagging system for tagging structured query language commands, the system comprising:

a parsing unit for parsing a structured query language string into components;

an analysis unit for analyzing the components and applying associated tags to the components; and

a string generation unit for concatenating the components with associated tags into a new string.

2. The system as claimed in claim 1, wherein the tags are extensible markup language tags.

3. The system as claimed in claim 1, further comprising a repository for holding a list of tags associated with structured query language components.

4. A method of tagging structured query language commands, the method comprising the steps of:

parsing an incoming structured query language statement and producing an input tree representation; and

traversing the input tree representation while generating a structured query language string with embedded extensible markup language tags.

5. A method of tagging structured query language commands, the method comprising the steps of:

parsing a structured query language string into components;

analyzing the components and applying associated tags to the components; and

concatenating the components with associated tags into a new string.

6. The method as claimed in claim 5, wherein the tags are extensible markup language tags.

7. The method as claimed in claim 5, wherein the analyzing step comprises the steps of:

comparing a component with a collection of tags; and

determining which tags is associated with the component.

8. The method as claimed in claim 5, wherein the step of applying comprises the step of adding an associated tag to the component.

9. A WITH clause transformation system comprising:

a WITH clause analysis module for analysing structured query language / online analytical programming (SQL/OLAP) windowed aggregates that are not supported by a target database system; and

a WITH clause transformation module for transforming SQL/OLAP windowed aggregates into semantically equivalent standard aggregate functions that are supported by the target database system.

10. A method of WITH clause transformation, the method comprising the steps of:

analysing a query containing SQL/OLAP windowed aggregates that are not supported by a target database system; and

transforming the query into a semantically equivalent query, having a WITH clause, that is supported by the target database system.

11. A parallel detail join system for merging elements in a plurality of tables, the system comprising:

a parallel detail analysis module for obtaining related elements in a plurality of tables; and

a parallel detail transformation module for merging the related elements in a singe table, the related elements merged such that the number of rows of a single element equals a largest cardinality of the single element and the number of columns equals the number of tables.

12. A method of merging elements in a plurality of tables, the method comprising the steps of:

obtaining related elements in a plurality of tables; and

merging the elements are merged into a single table such that the number of rows of a single element equals a largest cardinality of the single element and the number of columns equals the number of tables.

13. A method of merging elements in a plurality of tables, the method comprising the steps of:

analyzing elements in a plurality of tables;

creating a single table having a number of columns equal to the number of tables plus one, and a number of rows equal to the total of the largest cardinality of the elements in each table; and

placing the properties of each respective element in the single table in a number of rows equal to a largest cardinality of the respective element.

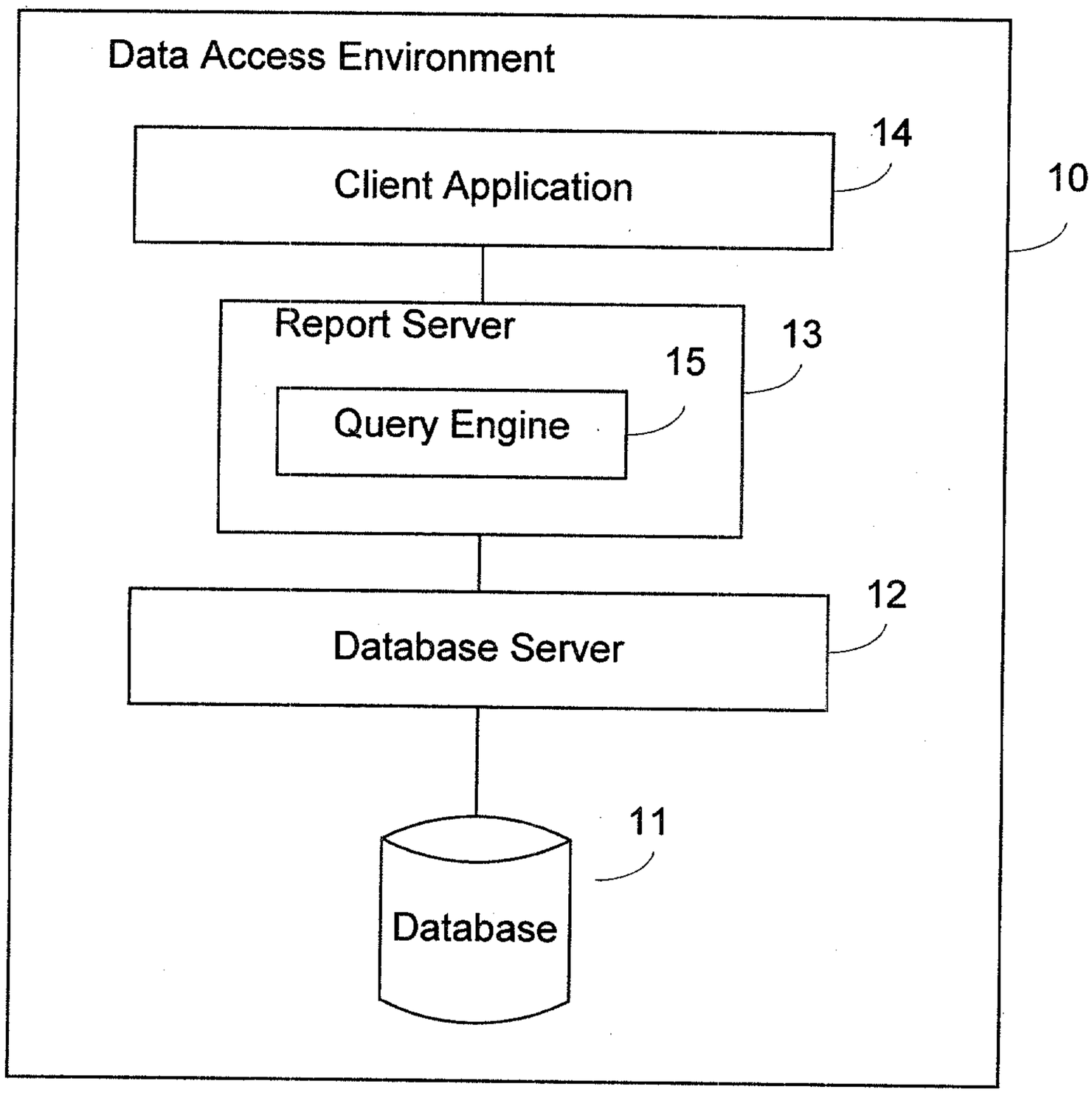

Data Access Environment
Client Application
14
10
Report Server
13
15
Query Engine
Database Server
12
11
Database

Figure 1

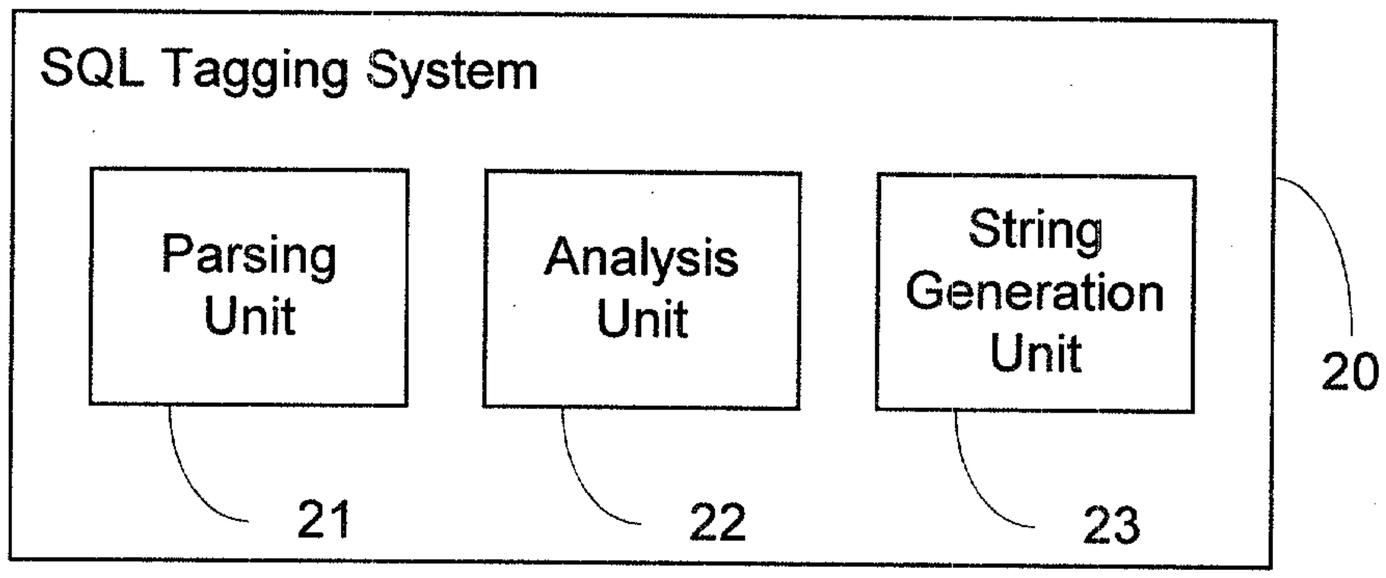


Figure 2

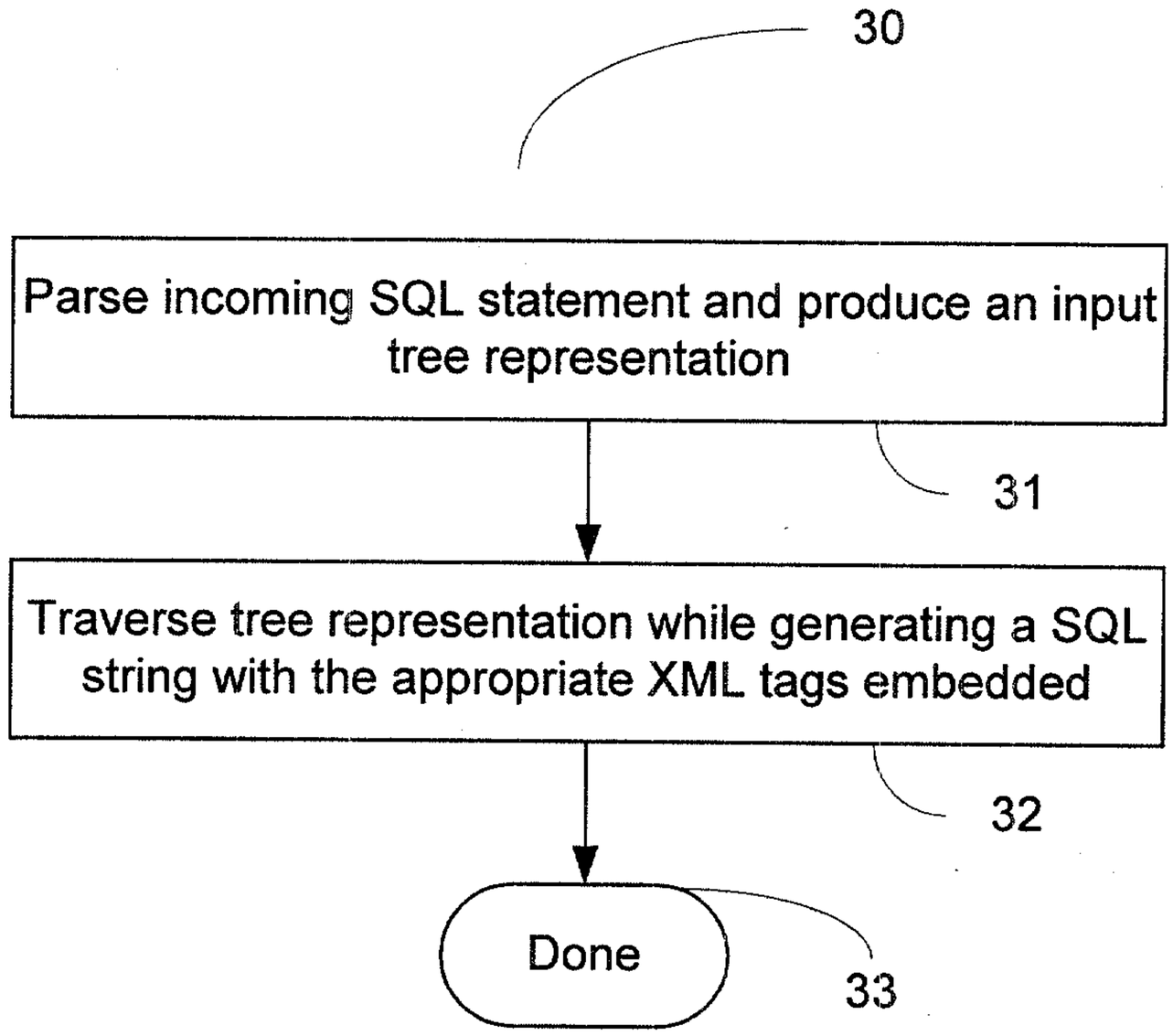


Figure 3

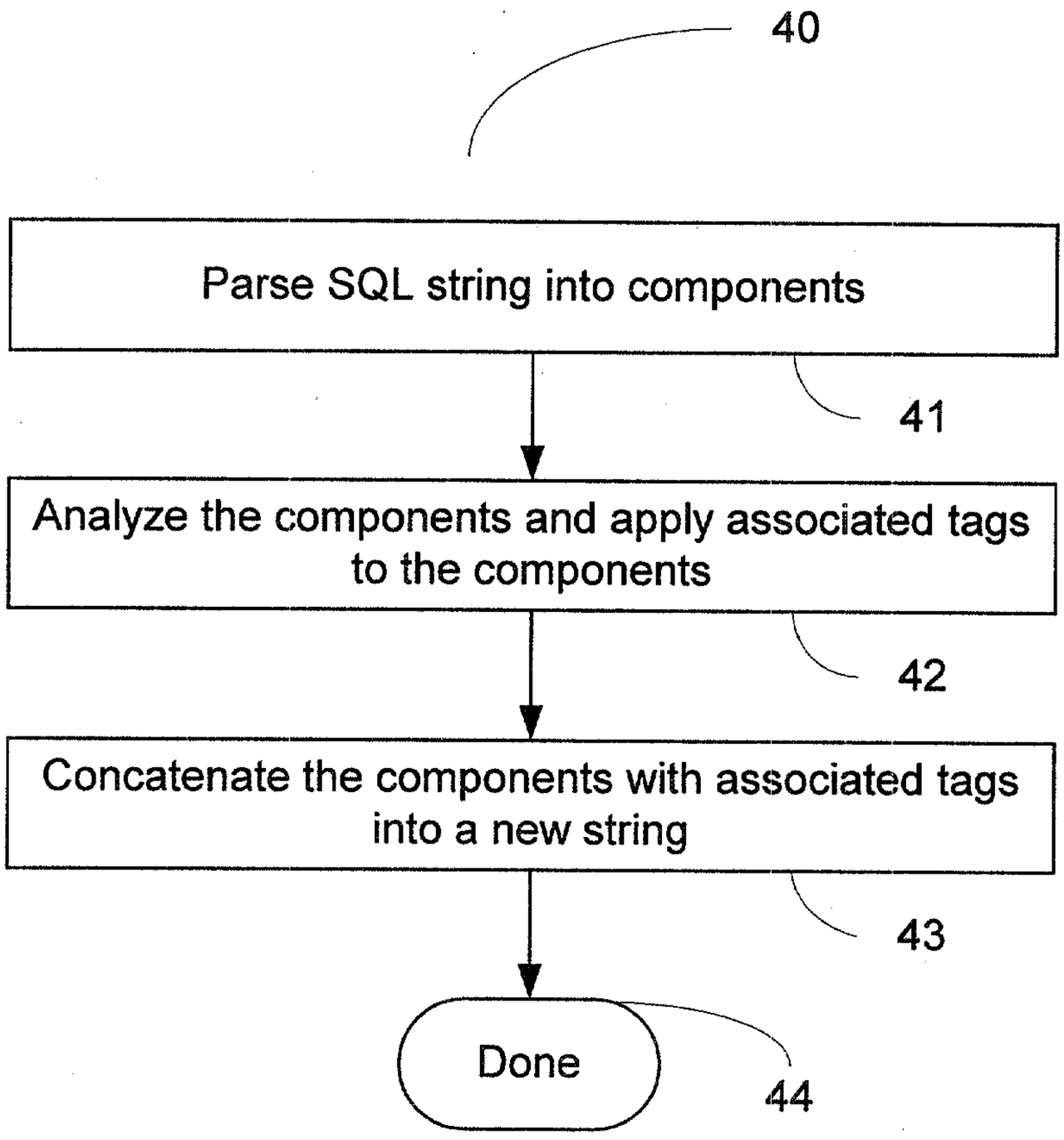

40
Parse SQL string into components
41
Analyze the components and apply associated tags to the components
42
Concatenate the components with associated tags into a new string
43
Done
44

Figure 4

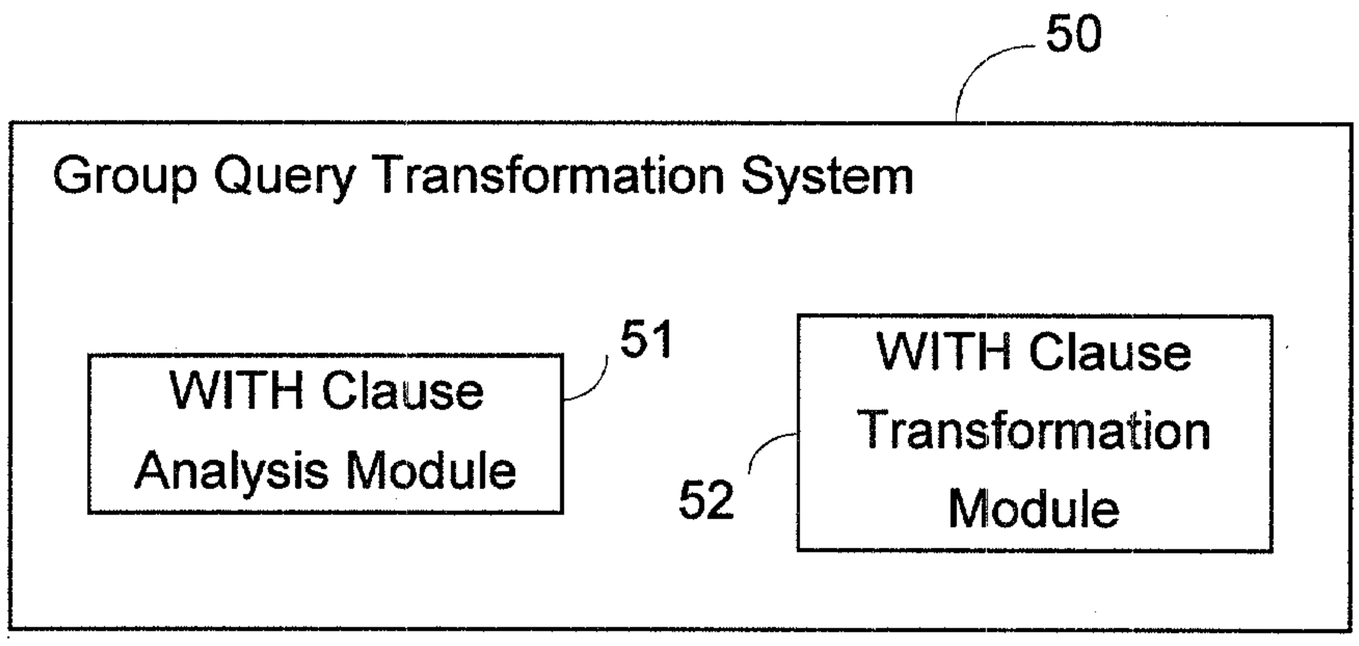


Figure 5

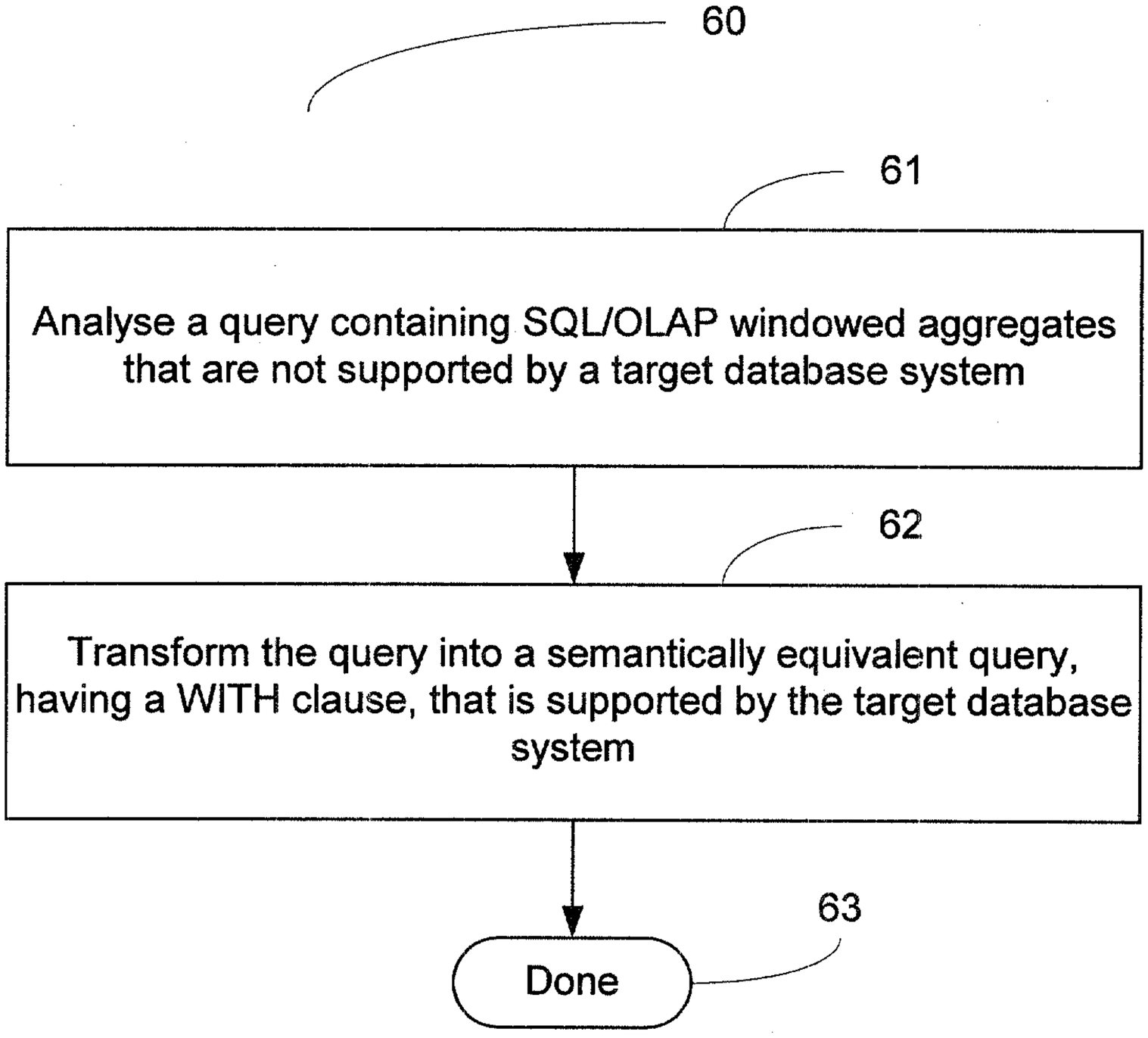


Figure 6

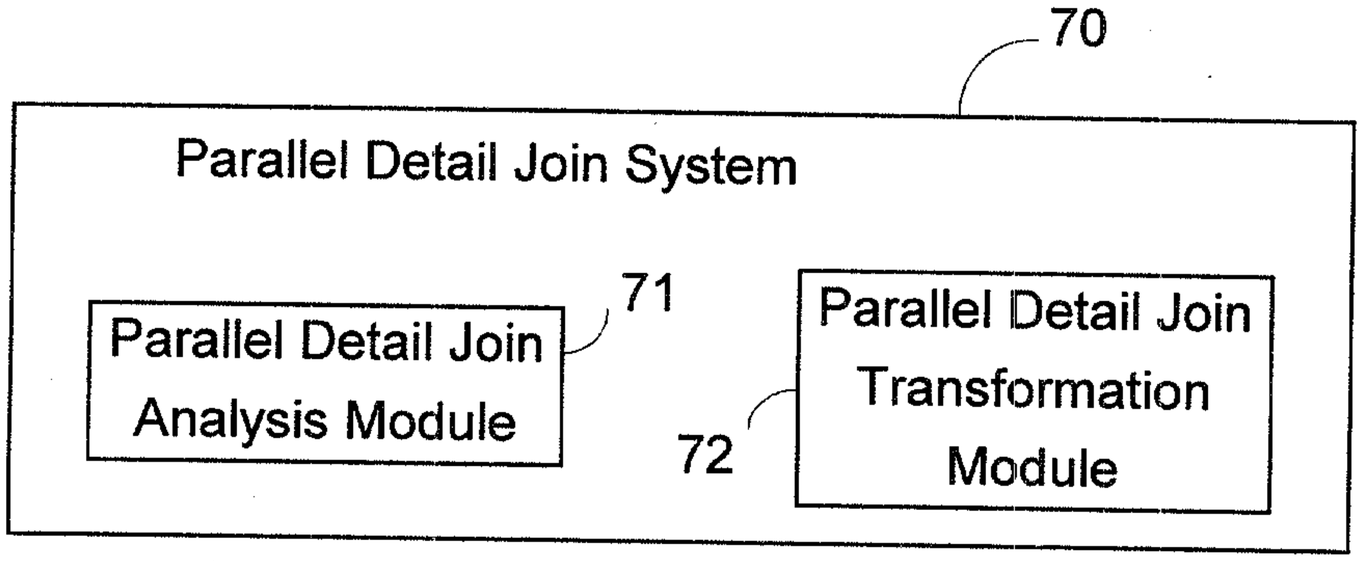


Figure 7

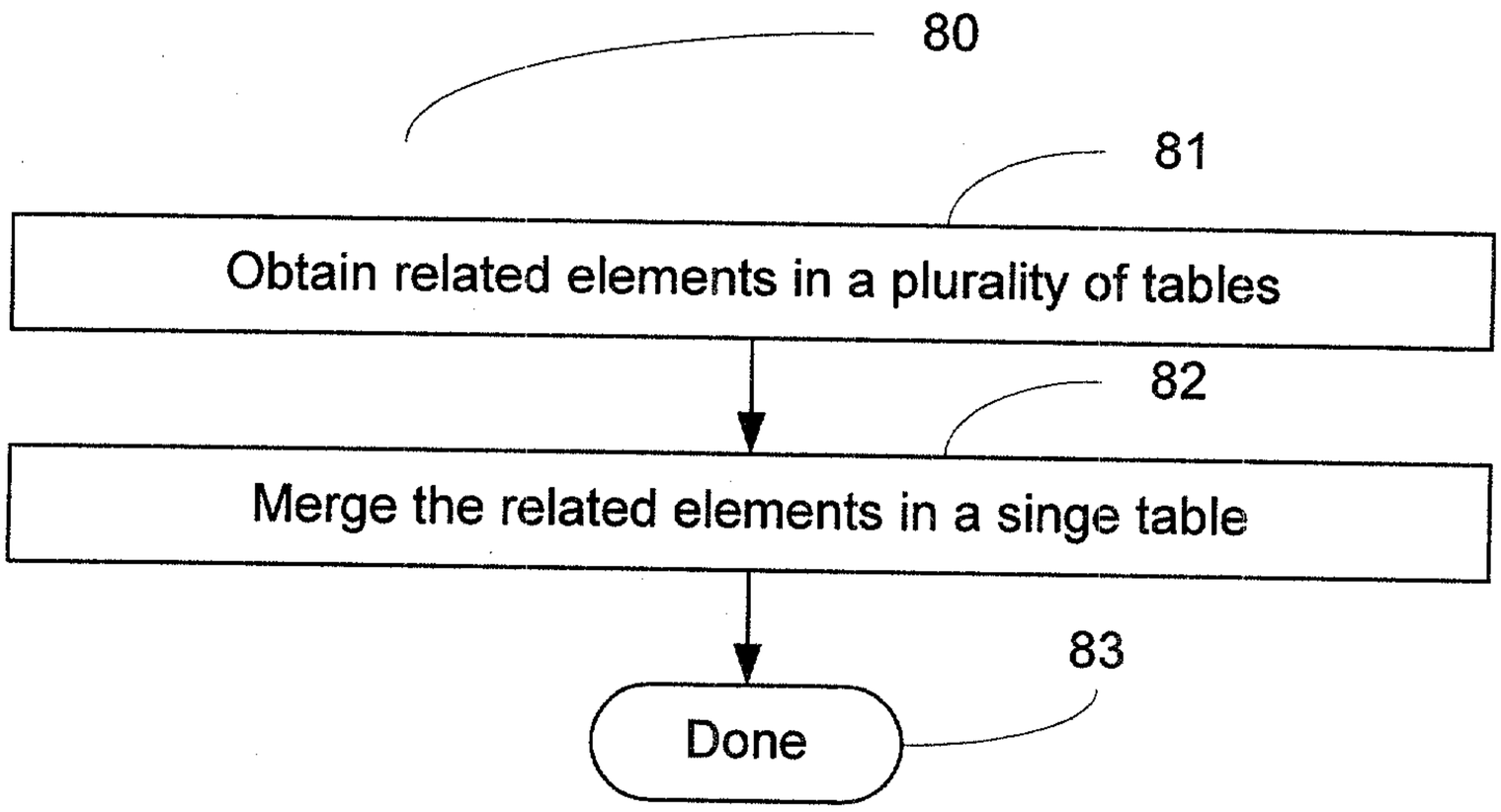


Figure 8

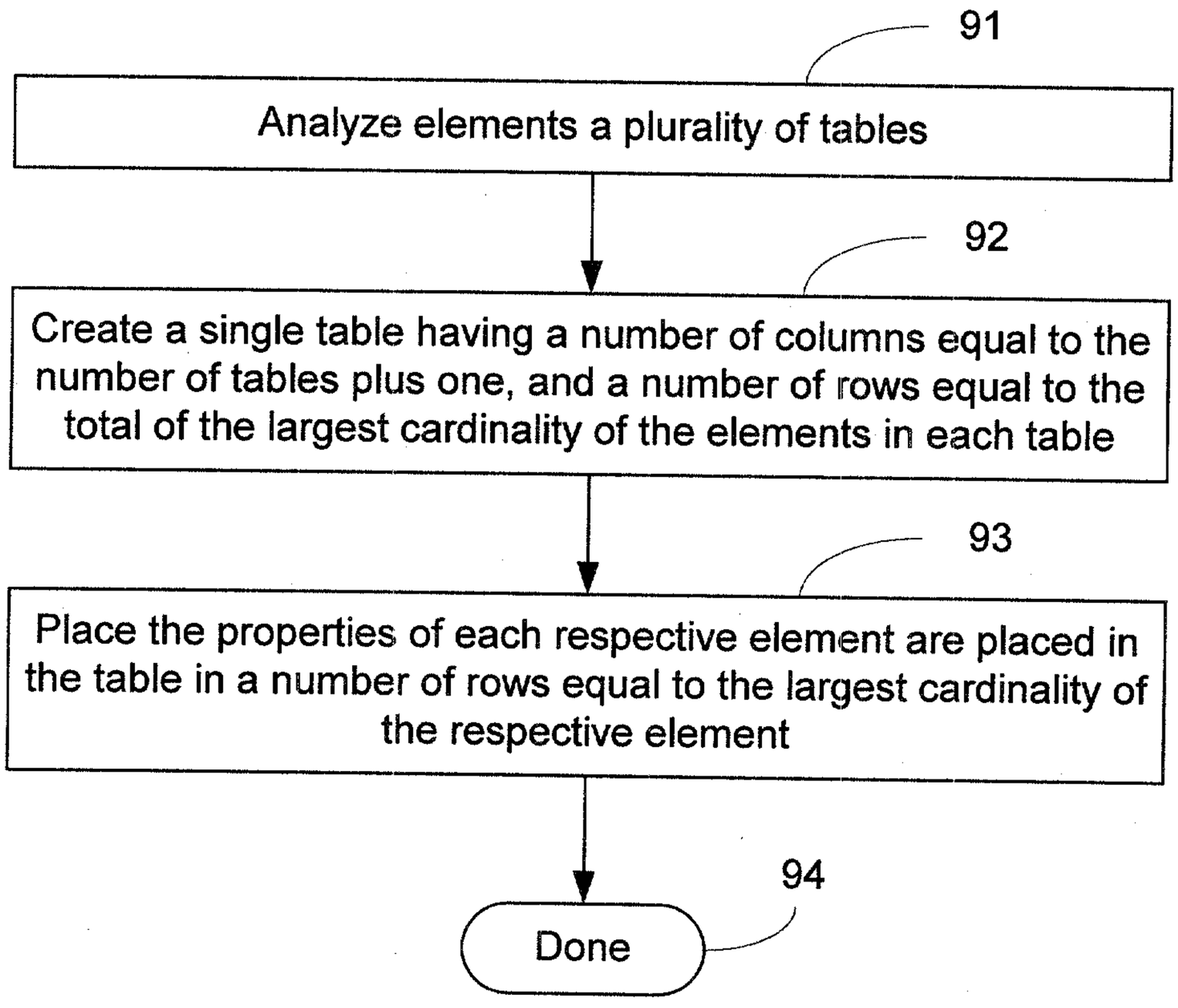
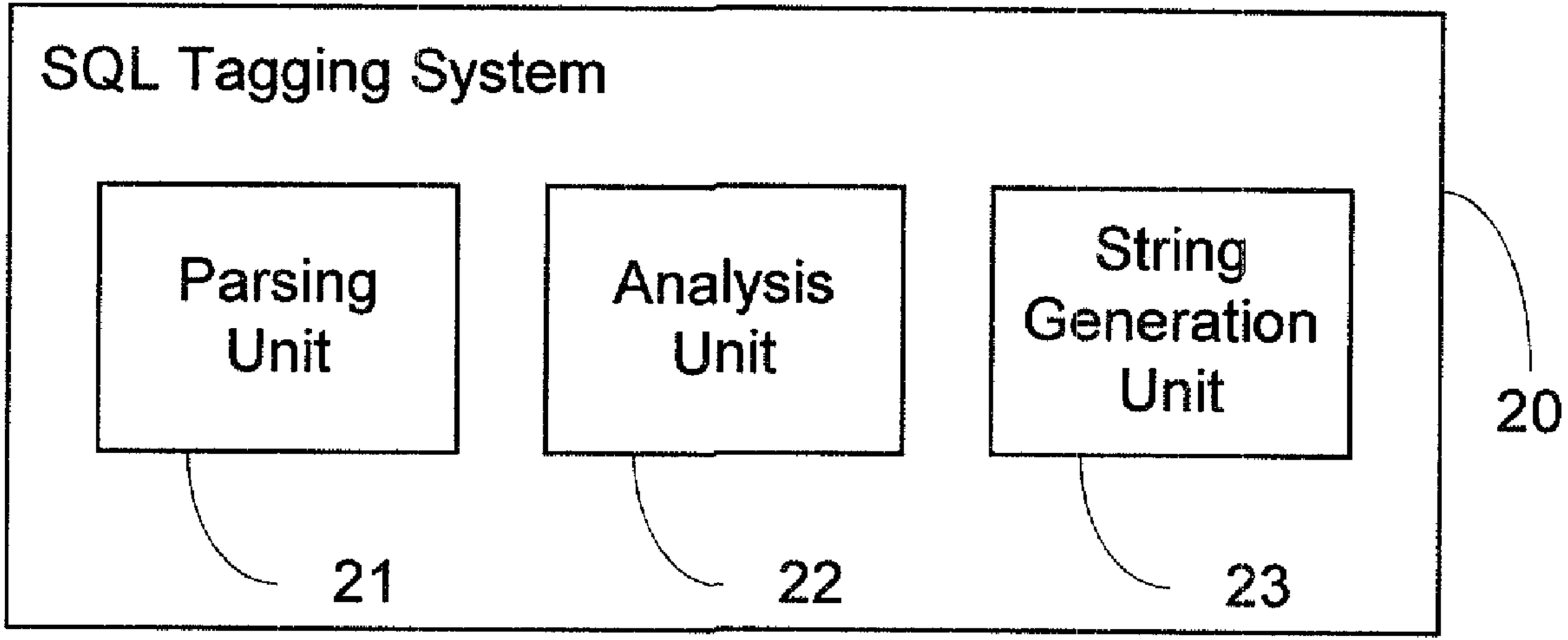

91
Analyze elements a plurality of tables
92
Create a single table having a number of columns equal to the number of tables plus one, and a number of rows equal to the total of the largest cardinality of the elements in each table
93
Place the properties of each respective element are placed in the table in a number of rows equal to the largest cardinality of the respective element
94
Done

Figure 9


SQL Tagging System
Parsing Unit
Analysis Unit
String Generation Unit
20
21
22
23