



G00002

D E S C R I P T I O N

Method for processing information

BACKGROUND OF THE INVENTION

Technical Field:

The present invention generally relates to a method for processing information and particularly to a method for processing structured information to be stored to a database and to a method for transferring such information to a database.

Description of the Related Art

Very often a certain amount of information originating from a party A has to be read and processed by a different party B. Such information can exist in a big variety of forms, it can contain numbers, words, sentences and the like in different variations such as free text, tables, and so on. In modern times such information comes up in large quantities and needs to be processed and understood by the receiving party quickly and reliably. Such information is often fed into databases on the receiving side of the data transfer. Someone who receives such information from a plurality of different sites and who has stored such information into a large database system, first has to read first the information from the database and subsequently has to parse the stored "packages" or -more generally - amounts of information. Then he is able to understand the information. Without such step of parsing the information the stored information has no real value except



166166

that of being stored on a data carrier. Those information packages are known as BLOBS (=Binary Large Object). Such BLOBS have to be parsed and thus their structure has to be determined found out before they can be read and the information contained in the BLOBS can be understood by anyone.

A problem comes up when it is necessary to read the contents of many BLOBS and to evaluate them in a similar way as it is done with the contents of databases. Such evaluations comprise queries which selectively access certain contents of a database and the results of which are lists of the queried information in a detailed form and sorted by any individual criterion. When a plurality of BLOBS is stored in a database, with prior art techniques an individual parsing process has to be done for each BLOB and it is not possible to direct a query immediately to a plurality of BLOBS without parsing them before. So any evaluation concerning a large amount of stored information is a lot of work and costs a lot of time although the information is stored in a electronically readable form.

In view of the above, it should be apparent that a need exists for a method to process large amounts of structured information to be stored in a database of a computer system more comfortably and more quickly.

SUMMARY OF THE INVENTION

It is therefore one object of the present invention to provide an improved method to process structured information to be stored in a database of a computer system.

It is another object of the present invention to provide an improved method for transferring information from an external site into a database.

It is yet another object of the invention to provide a computer system on which a database management system is installed which is configured to run a computer program implementing the processing method according to the invention.

The main idea for achieving these objects is to deduce directly database schema information from a given grammar characterizing the structure of the file contents to be transferred and to create a database schema directly out of the grammar. Said schema information is deduced while parsing with the help of the grammar. Mapping rules, which tell how to map contents adhering to a given grammar are deduced at the same time as the schema information is deduced.

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself however, as well as a preferred mode of use, further objects and advantages thereof, will best be understood by reference to the following detailed description of illustrative embodiments when read in conjunction with the accompanying drawings, wherein:

Figure 1 depicts a schematic flow chart of the processing method of the present invention,

Figure 2 is a comprehensive schematic sketch showing basic aspects of a first embodiment of the processing method of the present invention,

Figure 3 is a comprehensive schematic sketch showing basic aspects of a second embodiment of the processing method of the present invention,

Figure 4 is a schematic block diagram of the method of transferring information according to the present invention,

Figure 5 is a comprehensive schematic sketch showing the information transfer according to method of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

With general reference now to the figures and with special reference now to **Fig. 1** a certain amount of structured information 100 will now be explained as being object to the processing method of the present invention. The steps to be described handling said amount of information are parts of this method and are advantageously performed partly at the sending site computer system and partly at the receiving site computer system. Those computer systems are equipped with programs and tools in order to be able to perform said steps.

The information 100 is contained in an electronical (computer readable) file.

Grammars are determined -step 110 - to the information amount 100 by using for example naming conventions like certain file extensions or by side information.

A database schema - here for a relational database - corresponding to the grammars is deduced from the grammars as is depicted in step 120.

Next, in a further step 130, mapping rules are generated which determine how to map the structured contents of the information described by the grammars to a plurality of tables comprised of the database schema.

Finally, step 140, the database can be filled with any information corresponding to the database schema and to the grammar rules established before.

As a result, with this 'grammar based mapping' the information is stored in a structured form in the tables of the relational database system. As an advantage of the method described above according to the present invention detailed queries can be prepared by an user of the database which query the contents of the database and which bring up all information desired in sorted form and easy to understand because tables of any database are generally semantic unities. Thus, parsing the information needs not to be a part of the read access to the database, as it was in the prior art technique.

With reference now to Fig. 2, in a first simple embodiment of this processing method the so-called 'simple grammar based mapping' is described.

With simple grammar based mapping the complete schema 10 of a database 12 is deduced and generated from the grammar 14 which describes the structured information file contents 16. Mapping rules 18, which say how to map content adhering to the grammar to the database described by the deduced schema are statically deduced by the grammar, too.

The following grammar example is merely one possibility of numerous applications of this technique. It is a simple grammar for describing nestable record structures e.g. for a special computer language.

```
RECORD==>IDENTIFIER? 'STRUCT' MEMBER* 'END'  
MEMBER==>IDENTIFIER TYPE  
TYPE==>'STRING' 'INTEGER' RECORD  
IDENTIFIER==>('a'-'z' 'A'-'Z')+
```

This exemplary grammar shows that a record starts with an optional name of the structure followed by the keyword 'STRUCT' and by zero or more MEMBERS and ends with the keyword 'END'. A MEMBER is composed of an IDENTIFIER followed by a TYPE. An IDENTIFIER in turn is a concatenation of one or more letters and a TYPE is either one of the keywords 'STRING' or 'INTEGER' or is in turn a nested record.

A possible relational schema deduced from this grammar could be:

```
Table 'RECORD_TABLE' with columns:
  'record_key' of type integer which is the primary key of this table,
  'record_name' of type varchar,
  'number_of_members' of type integer.
Table 'MEMBER_TABLE' with columns:
  'member_key' of type integer which is the primary key of this table,
  'member_name' of type varchar,
  'parent_record_key' of type integer,
  'simple_type' of type integer,
  'complex_type' of type integer which is a foreign key to table
  'RECORD_TABLE'
```

The corresponding mapping rules could be:

```
IF the 'RECORD' rule of the grammar applies THEN
  insert a new row into the table RECORD_TABLE and generate a
  unique number identifying the record and put this number into
  the record_key field of the row. Then put the name of the
  record, if available, and the number of actual MEMBERS into
  the columns record_name and number_of_members of the row. For
  all MEMBERS update the corresponding rows by updating the
  field parent_record_key with the record_key value of this
  record.
ENDIF
```

```
IF the MEMBER rule of the grammar applies THEN
  insert a new row into the table MEMBER_TABLE and generate a
  unique number within this table and put it into the member_key
  field of the row. Put the value of 'IDENTIFIER' into
```

member_name of the row. If the type of this member is 'STRING' put 0 into the simple_type field and NULL into the complex_type field. If the type is 'INTEGER' put 1 into the simple_type field and NULL into the complex_type field. If the type is in turn a record structure then put the record_key value of its corresponding database entry into the complex_type field and NULL into the simple_type field.

ENDIF

The next example demonstrates the mapping described above. The following content describes personal data and its structure adheres to the given grammar.

```

PersonData  STRUCT
              Name  STRING
              Age   INTEGER
              Address STRUCT
                  Street STRING
                  Zip   STRING
                  City   STRING
                  Country STRING
              END
            END

```

With the mapping rules from above the result of the grammar based mapping could be:

RECORD_TABLE

record_key	record_name	number_of_members
1	PersonData	3
2	NULL	4

MEMBER_TABLE

member_key,	member_name,	parent_record_key,	simple_Type,	complex_Type
1	Name	1	0	NULL
2	Age	1	1	NULL
3	Address	1	NULL	2
4	Street	2	0	NULL
5	Zip	2	0	NULL
6	City	2	0	NULL
7	Country	2	0	NULL

So, when a database is filled with a plurality of different amounts or packages of information which have a structure able to be generalized and described by the grammar 14 from above a comfortable read or query from the database including complete state-of-the-art queries can be performed without having to parse the same plurality of BLOBs first what would correspond to prior art.

In a second embodiment of the processing method of the present invention outlined in **Fig. 3** the so-called 'complex grammar based mapping' is described.

With complex grammar based mapping the schema 10 is not necessarily deduced completely by the grammar 14, but both, by the grammar and a so-called meta-context 20, when a certain rule of the grammar is applied to a content. The schema 10 may partially be deduced from the grammar and during parsing both schema extensions and mapping of information into the database can occur.

To describe that in more detail, in the 'record' - example described above it could be desirable by the context to generate for each record its own table and to register each generated table into a registration table. The context would then reflect e.g. certain user or application needs.

Which tables will be created is only known at parse time. In this example just one table is statically deduced from the grammar:

RECORD_REGISTRY_TABLE contains all records for which tables exist.

Table 'RECORD_REGISTRY_TABLE' with column 'record_table_name' which is the primary key of this table.

Compared to the 'simple grammar based mapping' the rules thus can also contain schema extension information, e.g. rules which generate additional database tables or create additional rows or fields in already created tables, etc.

For this example the set of rules can be:

```
IF the 'RECORD' rule of the grammar applies THEN
create a new table into the schema. The name of the table is
given by the name of the parsed record. The columns of the
table are (member_name as varchar, simple_type as integer,
complex_type as varchar). member_name is the primary key. Put
the record name (value of IDENTIFIER) into the
RECORD_REGISTRY_TABLE table. If no record name is given create
a unique one.
```

```
ENDIF
```

```
IF the MEMBER rule of the grammar applies THEN
insert a new row into the table corresponding to the record to
which this member belongs. Put the value of IDENTIFIER into
member_name of the row. If the type of this member is 'STRING'
put 0 into the simple_type field and NULL into the
complex_type field. If the type is 'INTEGER' put 1 into the
simple_type field and NULL into the complex_type field. If the
type is in turn a record structure then put the record name
into the complex_type field and NULL into the simple_type
field.
```

```
ENDIF
```

When parsing the following example:

```

PersonData  STRUCT
              Name      STRING
              Age        INTEGER
              Address     STRUCT
                        Street  STRING
                        Zip     STRING
                        City    STRING
                        Country  STRING
              END
            END

```

the schema may be extended with the tables 'PersonData' and 'SubRecord1'; since no record name is given for the nested record 'SubRecord1' will be created as its name:

Table 'PersonData' with columns:

- 'member_name' of type varchar which is the primary key of this table,
- 'simple_type' of type integer,
- 'complex_type' of type varchar

Table 'SubRecord1' with columns

- 'member_name' of type varchar which is the primary key of this table,
- 'simple_type' of type integer,
- 'complex_type' of type varchar

After parsing the tables will contain the following data:

RECORD_REGISTRY_TABLE

(record_table_name)

PersonData

SubRecord1

PersonData

member_name, simple_type, complex_type

Name 1 NULL

Age 2 NULL

Address NULL SubRecord1

SubRecord1

member_name, simple_type, complex_type

Street 1 NULL

Zip 1 NULL

City 1 NULL

Country 1 NULL

As described in the first embodiment above a comfortable read or query from the database can be performed without parsing BLOBs before.

With reference now to **Fig. 4** and **Fig. 5** an example is given for the application of the processing method according to the present invention in which a direct read/query access to the database 12 without parsing the information before can be performed:

A receiving site party B (right side in Fig. 5) is going to receive a plurality of same type information packages, i.e. information packages which have a common general structure, from different parties of which one - party A - is depicted on the left side in Fig. 5.

B could be a hospital's database receiving from a doctor A's practice computer a certain amount of patient data. For each of A's patients the same type of data , i.e. information package, is expected to be transferred.

Party A designs or uses - 400 - one or a set of so called DTDs (document type declarations) comprising the grammar rules concerning and characterizing the contents of the information amounts to be transferred.

Next, in step 410 one of the DTD files 30 are sent to B. This can happen in any electronic form or in form of a paper being sent as letter from A to B.

Next, in step 420 B receives the DTD file 30 and generates a database schema 10 according to the information contained in the DTD file.

24 MAY 2002

As can be seen in the upper part of the drawing in Fig. 5, too, A sends the DTD file of step 410, Fig. 4 to B. As can be seen from the database symbol 12, a database schema 10 is derived from the DTD file content and is generated dynamically on receipt of the DTD file as it is depicted with the four 'dummy' points in the database 12. When the database schema 10 is completed and the computer system on which the database 12 is residing is ready to receive data, the data transfer can begin.

In the next row of Fig. 5 an information package is sent from A to B. The information is split up into the four tables representing an exemplary database schema of the receiving site B. Thus, all information is stored in it's correct place in the database. This send process is repeated now a certain number of times and the amount of information, stored in B's database is getting larger and larger. A third party C is depicted to send the same type of information to B's database, too.

As is depicted in the bottom part right side of Fig. 5 the total of stored information can be accessed with state-of-the-art queries selecting the desired table contents from B's database. Parsing of BLOBS is not necessary as it was in the state of the art technology.

Back to Fig. 4, in step 430, A sends a certain amount of information - a plurality of information packages like files, e-mails, etc. according to the grammar rules and the database schema being setup in B's computer system. This is depicted as second 'row' in Fig. 5.

In the next step 440 B's computer system receives that information amount and stores it automatically in database tables 32 according to the database schema 10. As can be seen

from the loop depicted in Fig. 4, A's send process can be repeated now as often as required. Or, another party C - another doctor's practice - can send further amounts of information of the same type which is depicted in the third row in Fig. 5. Then, the latter information will be stored in the same way since the information is structured in the same way as A's information.

As a result, the receiving site database 12 is filled up with information correctly structured and stored in the corresponding database tables 32. Thus, individual queries can be setup by B's staff immediately and comfortably over the whole quantity of transferred information packages and no BLOB parsing is necessary when the data is read from the database. Thus, the amount of time necessary to read and understand the data from the database is minimized.

Next, the steps 120, 130 of Fig. 1 are described in more detail with special reference to so called context free grammars which form the main application part of the method of the present invention. The structure of the file contents - or more generally expressed - the structure of any information package or amount in the wording of the present application is often described by context free grammars. Here, the non-terminals of the grammar are mapped to tables of a relational database system. This is a natural mapping because non-terminals are normally used to describe semantic entity groups syntactically. These entity groups can in turn be mapped to tables.

Without loss of generality it is assumed that the rules for context free grammars are of the form:

```
[1]      ls ==> rs1 rs2 ... rsN
or
[2]      ls ==> rs1  rs2  ...  rsN
```

or
[3] $ls \Rightarrow (rs)^+$
or
[4] $ls \Rightarrow (rs)^*$

ls stands for 'left side' and rs stands for 'right side'. ls ever represents a non-terminal and rs represents either a terminal, empty-terminal or a non-terminal.

[1] is called concatenation of structures and means that the left side is composed by the concatenation by one or more terminals or non-terminals of the right side.

[2] is called selection of structures and means that the left side is composed exactly of one terminal or non-terminal on the right side.

[3] is called one-repetition of a structure and means that the left side is composed by one or more repetitions of rs.

[4] is called zero-repetition of a structure and means that the left side is composed by zero or more repetitions of rs.

Each context free grammar can be described with a finite set of rules each adhering to one of these four patterns.

Furthermore, it can be assumed without loss of generality that each non-terminal appears exactly once on the left-side of the rules. Regular expressions are a special subclass of context free grammars and thus can be described by a set of rules adhering these patterns. In practice non-terminals are split into two groups: token-classes and real non-terminals. The former ones are recognized by lexical analyser and the latter ones by parsers. In the present examples these groups are distinguished with quotes for terminals like 'Name', 'A', '0', ':', with <>.

24 MAY 2002

for token-classes like <STRING>, <IDENTIFIER>, <NUMBER> and capitalized words without any pre- or postfix for non-terminals like ADDRESS, EMPLOYEE_NUMBER.

'Real non-terminals' are referred to as non-terminals.

A 'simple grammar based mapping' of these rules could be:

For each non-terminal a table is created. For each table automatically a primary key will be created. This is simply a number which uniquely identifies a row. This key is automatically incremented for each new row which is inserted into the table.

For each rule which matches pattern [1] columns which represent the foreign key of each non-terminal on the right side of the rule are added to the table corresponding to the non-terminal on the left side of the rule. For each token-class on the right side of the rules a column of type varchar is added. If the rule is matched at parse time then for each non-terminal on the right side of the rule the foreign key is put into the appropriate columns and for each token-class the concrete value is put in the appropriate column.

E.G. the grammar rules

EMPLOYEE	==>	'Name: ' NAME 'No: ' <NUMBER>
NAME	==>	FIRST_NAME LAST_NAME
FIRST_NAME	==>	<STRING>
LAST_NAME	==>	<STRING>
STRING	==>	('a'-'z''A'-'Z')+
NUMBER	==>	('0'-'9')+

results in following schema extension:

Table 'EMPLOYEE' with columns:

- 'employee_key' of type integer which is primary key of this table,
- 'name' of type varchar which is a foreign key to table 'NAME',
- 'employee_number' of type varchar.

Table 'NAME' with columns:

- 'name_key' of type integer which is primary key of this table,
- 'first_name' of type varchar,
- 'last_name' of type varchar

After parsing the strings: 'Name: John Miller No: 000007',
'Name: Liz Parker No: 000011' and 'Name: Cindy Smith No:
000003' the following entries will be generated:

EMPLOYEE

employee_key,	name,	employee_number
1	1	'000007'
2	2	'000011'
3	3	'000003'

NAME

name_key,	first_name,	last_name
1	'John'	'Mueller'
2	'Liz'	'Parker'
3	'Cindy'	'Smith'

It should be noted that the value in column 'name' of table EMPLOYEE refers to a row in table NAME.

For each rule which matches pattern [2] columns which represent the foreign key of each non-terminal on the right side of the rule are added to the table corresponding to the non-terminal on the left side of the rule. For each token-class a column with type varchar is generated and if there are terminals on the right side then a column of type varchar with name 'terminal_value' is generated. If the rule is matched at parse time with a non-terminal then for the matched non-terminal on the right side of the rule the foreign key is put into the appropriate column and all other columns are set to NULL. If it is matched with a token-class then the concrete value is put into the appropriate column and all other columns are set to NULL. If a terminal matches on the

24 MAY 2002

right side then the value of the terminal is put into the 'terminal' column and all other columns are set to NULL.

In a further exemplary potential file content the grammar rules

```
LIVING_THING ==> ANIMAL PLANT
ANIMAL        ==> 'A:' <STRING>
PLANT         ==> 'P:' <STRING>
STRING        ==> ('a'-'z' 'A'-'Z')+
```

results in following schema extension:

Table 'LIVING_THING' with columns

- 'living_think_key' of type integer which is primary key of this table,
- 'animal' of type varchar which is foreign key to table 'ANIMAL',
- 'plant' of type varchar which is foreign key to table 'PLANT'

Table 'ANIMAL' with columns

- 'animal_key' of type integer which is primary key of this table,
- 'animal_name' of type varchar

Table 'PLANT' with columns.

- 'plant_key' of type integer which is primary key of this table,
- 'plant_name' of type varchar

When parsing the strings: 'A: Dog', 'P: Oak', 'P: Alder' and 'A: Squirrel', then the following entries will be generated:

LIVING_THING		
living_think_key,	animal,	plant
1	1	NULL
2	NULL	1
3	NULL	2
4	2	NULL

ANIMAL	
animal_key,	animal_name
1	'Dog'
2	'Squirrel'

PLANT	
plant_key,	plant_name
1	'Oak'
2	'Alder'

A further example is given for complete understanding. In this example a customer can be identified by name or by customer

number. The subject of an order can be a table, a chair or something else which will be corresponding to the any_other_subject column of the SUBJECT table. The grammar rules

```

ORDER      ==> 'Order from: ' CUSTOMER ' Subject is: 'SUBJECT
SUBJECT    ==> 'Table' 'Chair' <STRING>
CUSTOMER   ==> NAME <NUMBER>
NAME       ==> FIRST_NAME LAST_NAME
FIRST_NAME ==> <STRING>
LAST_NAME  ==> <STRING>
STRING     ==> ('a'-'z'"A'-'Z')+
NUMBER     ==> ('0'-'9')+

```

results in following schema extension:

Table 'ORDER' with columns

- 'order_key' of type integer which is primary key of this table,
- 'customer' of type integer which is foreign key to table 'CUSTOMER',
- 'subject' of type integer which is foreign key to table 'SUBJECT'.

Table 'SUBJECT' with columns

- 'subject_key' of type integer which is primary key of this table,
- 'terminal' of type varchar,
- 'any_other_subject' of type varchar

Table 'CUSTOMER' with columns

- 'customer_key' of type integer which is primary key of this table,
- 'name' of type varchar which is foreign key to table 'NAME',
- 'number' of type varchar

Table 'NAME' with columns

- 'name_key' of type integer which is primary key of this table,
- 'first_name' of type varchar,
- 'last_name' of type varchar.

When parsing the strings: 'Order from: John Miller Subject is: Table', 'Order from: 000918 Subject is: Water', 'Order from: Liz Parker Subject is: Car' and 'Order from: Cindy Smith Subject is: Chair' then the following entries will be generated:

ORDER		
order_key,	customer,	subject
1	1	1
2	2	2
3	3	3
4	4	4
SUBJECT		
subject_key,	terminal,	any_other_subject
1	'Table'	NULL
2	NULL	'Water'
3	NULL	'Car'
4	'Chair'	NULL
CUSTOMER		
customer_key,	name,	number
1	1	NULL
2	NULL	'000918'
3	2	NULL
4	3	NULL
NAME		
name_key,	first_name,	last_name
1	'John'	'Miller'
2	'Liz'	'Parker'
3	'Cindy'	'Smith'

For each rule which matches pattern [3] and has a non-terminal on the right side a column for primary key purposes is added to table corresponding to the non-terminal of the left side of the rule. Also a column for counting purposes is added to this table which also serves as a part of the primary key. For the non-terminal on the right side a column for the foreign key is added to the table. If the rule is matched at parse time then for each instantiation of the right side a row with the sequence number of the item and the foreign key is created and inserted into the table.

E.G. the grammar rules

```

LIST_OF_NAMES ==> (NAME)+
NAME          ==> FIRST_NAME LAST_NAME
FIRST_NAME   ==> <STRING>
LAST_NAME    ==> <STRING>
STRING       ==> ('a'-'z' 'A'-'Z')+

```

results in following schema extension:

Table 'LIST_OF_NAMES' with columns:

'list_key' of type integer is part of the primary key,
 'sequence_number' of type integer is part of primary key,
 'name' of type varchar which is foreign key to table 'NAME'.

Table 'NAME' with columns:

'name_key' of type integer which is primary key of this table,
 'first_name' of type varchar,
 'last_name' of type varchar.

When parsing the strings: 'John Miller Liz Parker Cindy Smith'
 and 'Elvis Presley Sandra Brown' then the following entries
 will be generated:

LIST_OF_NAMES		
list_key,	sequence_number,	name
1	1	1
1	2	2
1	3	3
2	1	4
2	2	5

NAME		
name_key,	first_name,	last_name
1	'John'	'Miller'
2	'Liz'	'Parker'
3	'Cindy'	'Smith'
4	'Elvis'	'Presley'
5	'Sandra'	'Brown'

For each rule which matches pattern [3] and has a token-class on the right side a column for primary key purposes is added to table corresponding to the non-terminal of the left side of the rule. Also a column for counting purposes is added to this table, it also serves as a part of the primary key. For the token-class on the right side a column of type varchar is added to the table. If the rule applies during parse time then for each instantiation of the right side a row with the sequence number of the item and the concrete token-class value created and inserted into the table.

E.G. the grammar rules

```
LIST_OF_FIRST_NAMES ==> (<STRING>)+
STRING              ==> ('a'-'z''A'-'Z')+
```

result in following schema extension:

Table 'LIST_OF_FIRST_NAMES' with columns:

'list_key' of type integer is part of the primary key,
 'sequence_number' of type integer is part of the primary key,
 'first_name' of type varchar.

When parsing the strings: 'John Liz Cindy' and 'Elvis Sandra'
 then the following entries will be generated:

LIST_OF_FIRST_NAMES		
list_key,	sequence_number,	first_name
1	1	'John'
1	2	'Liz'
1	3	'Cindy'
2	1	'Elvis'
2	2	'Sandra'

For each rule which matches pattern [3] and has a terminal on the right side a column for primary key purposes is added to table corresponding to the non-terminal of the left side of the rule. Also a column for counting purposes is added to this table. If the rule applies during parse time then a row with the number of occurrences of the terminal is created and inserted into the table.

E.G. the grammar rule

NUMBER_OF_ZEROS==> ('0')+

results in following schema extension:

Table 'NUMBER_OF_ZEROS' with columns:

'list_key' of type integer is part of the primary key,
 'counter' of type integer.

When parsing the strings: '0000000000', '000' and '000000' then the following entries will be generated:

NUMBER_OF_ZEROS	
list_key,	counter
1	10
2	3
3	6

For each rule which matches pattern [4] the same as for [3] applies.

The processing and the transferring method, respectively, according to the present invention can be applied

advantageously in conjunction with XML (Extensible Markup Language) documents because each XML document refers optionally to a document type declaration file (DTD). As is known in the art a DTD uses among other constructs mainly regular expressions to describe the structure of an XML file. The following sample DTD is given to demonstrate the ability to apply a DTD written in XML to the processing and transferring method according to the present invention. The DTD contains the structure of a list of employees.

```
<?xml encoding="US-ASCII"?>

<!-- DTD filename is employee.dtd -->

<!ELEMENT list_of_employees (employee)+>
<!ELEMENT employee (name,email*,link?)>
<!ATTLIST employee id ID #REQUIRED>
<!ELEMENT name (family,given)>
<!ELEMENT family (#PCDATA)>
<!ELEMENT given (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT link EMPTY>
<!ATTLIST link
  manager IDREF #IMPLIED
  subordinates IDREFS #IMPLIED>
```

A XML document adhering to the DTD given above may look like:

24 MAY 2002

```
<!DOCTYPE employees SYSTEM "employees.dtd">

<list_of_employees>

  <employee id="007" >
    <name><family>Miller</family>
    <given>Liz</given></name>
    <email>Liz.Miller@de.ibm.com</email>
    <link subordinates="009 011"/>
  </employee>

  <employee id="009" >
    <name><family>Smith</family>
    <given>Bill</given></name>
    <email>CBillen@de.ibm.com</email>
    <link manager="007"/>
  </employee>

  <employee id="011">
    <name>
      <family>Cochran</family> <given>Ray</given>
    </name>
    <link manager="007"/>
  </employee>

</list_of_employees>
```

This XML document describes a list of employees with their names and their relation to each other. It has to be stressed that the structure of this XML document is given by the DTD and the DTD describes this structure grammar-like with regular expressions.

The rule of the DTD must be read as:

```
<!ELEMENT list_of_employees (employee)+>
```

The rule says that a list of employees contains one or more employee and that such a list have to be enclosed by the tags `<list_of_employees>` and `</list_of_employees>`.

```
<!ELEMENT employee (name,email*,link?)>
  <'ATTLIST employee id ID #REQUIRED>
```

These rules say that an employee have to contain a name, can contain several email addresses and can also have optional a link specified. Further must have employees a unique ID. An employee entry in the content file must be tagged with `<employee>` and `</employee>`.

```
<!ELEMENT name (family,given)>
<!ELEMENT family (#PCDATA)>
<!ELEMENT given (#PCDATA)>
```

These rules say that a name consists of family name and of given name, which in turn consist of an arbitrary character string. A name entry in the content file must be tagged with `<name>` and `</name>`. Family name and given name must be tagged with `<family>`, `</family>` and `<given>`, `</given>`.

```
<!ELEMENT email (#PCDATA)>
```

The rule says that an email consists of an arbitrary character string. An email entry must be tagged with `<email>` and `</email>`.

```
<!ELEMENT link EMPTY>
<!ATTLIST link
  manager IDREF #IMPLIED
  subordinates IDREFS #IMPLIED>
```

These rules say that a link has two optional attributes, one referring to the manager and the second referring to a list of subordinates.

Each `<!ELEMENT ...>` entry is called element type. The above example shows seven different element types:

`list_of_employees`, `employee`, `email`, `name`, `family`, `given` and `link`.

A document element adheres to the structure of a given element type and can have attributes (also defined in the DTD). These attributes are described by the `<!ATTLIST ...>` entry.

The methods of the present invention are now seen to be able to be applied to relational databases by describing the mapping of elements adhering to the element type definitions.

One simple implementation of this mapping technology could be to create for each element type a table in the database schema which is quite the same approach like for context free grammars.

The following tables can thus be generated:

```
LIST_OF_EMPLOYEES
EMPLOYEE
NAME
FAMILY
GIVEN
EMAIL
LINK
```

The columns of the tables correspond to the definition of the element type itself and to its attributes. In this example quite the same rules are used as for context free grammars, so the following table definitions are achieved:

24 MAY 2002

Table 'LIST_OF_EMPLOYEES' with columns:

'list_id' of type integer which is primary key,

'sequence_number' of type integer which is foreign key to 'EMPLOYEE'.

Table 'EMPLOYEE' with columns.

'employee_id' of type integer which is primary key,

'employeeid' of type varchar,

'name' of type integer which is foreign key to table 'NAME',

'email' of type integer which is foreign key to table 'EMAIL',

'link' of type integer which is foreign key to table 'LINK'.

Table 'NAME' with columns.

'name_id' of type integer which is primary key,

'family' of type integer which is foreign key to table 'FAMILY',

'given' of type integer which is foreign key to table 'GIVEN'.

Table 'FAMILY' with columns.

'family_id' of type integer which is primary key,

'name' of type varchar

Table 'GIVEN' with columns.

'given_id' of type integer which is primary key,

'name' of type varchar

Table 'EMAIL' with columns

'email_id' of type integer which is primary key,

'emailaddress' of type varchar

Table 'LINK' with columns

'link_id' of type integer which is part of the primary key,

'subordinate_number' of type integer.

'subordinate' of type integer which is foreign key to table 'EMPLOYEE',

'manager' of type integer which is foreign key to table 'EMPLOYEE'

In the foregoing specification the invention has been described with reference to a specific exemplary embodiment thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention as set forth in the appended claims. The specification and drawings are accordingly to be regarded as illustrative rather than in a restrictive sense.

The processing method according to the present invention is adaptable for object-oriented databases to store the structured information, too.

24 MAY 2002

LIST OF REFERENCE SIGNS

10	schema
12	database (db)
14	grammar
16	file contents
18	mapping rules
20	meta-context
30	DTD file
32	db tables
100	information package
110	determine grammars
120	deduce db schema from grammars
130	generate mapping rules from grammars
140	storing information to database according to schema and mapping rules
400	prototype information
410	parse information
420	generate DTD
430	send DTD to B
440	receive DTD and generate db schema
450	send plurality of information packages
460	receive information

24 MAY 2002

C L A I M S

1. A method to process structured information (100) to be stored to a database of a computer system, comprising the steps of

determining (110) the grammars describing the structure of information,

deducing (120) a database schema from said grammars,

generating (130) mapping rules from said grammars,

storing (140) the structured information to said database according to the schema and mapping rules.
2. The method according to claim 1, characterized by the step of

mapping non-terminals of a grammar to tables (32) of a relational database (12).
3. The method according to claim 2, characterized by the step of

using context free grammars to describe the structured information (100).
4. The method according to claim 3, characterized by the step of

using a document language to describe the structured information (100) and generating a document description file 30 reflecting the structured contents

of a plurality of information packages (100) and deducing said database schema (12) from said document description file (30).

5. The method according to claim 4, characterized by the step of

using XML to describe the structured information (100).

6. The method according to claim 5, characterized by the step of

mapping non-terminals of a grammar to structure elements of object oriented database systems.

7. A method of transferring an amount of information from an external site into a database, the amount of information comprising a plurality of at least similarly structured information packages (100) being able to be generalized by a common general information structure, said method comprising the steps of

generating once a document description file (30) (DTD) characterizing the general structure of the information packages (100) to be transferred,

generating once a database schema (10) in the receiving site database (10) according to the document description file (30),

sending said plurality of information packages (100) to the receiving site database (12), and

writing the information to the database (12)
corresponding to the schema (10).

8. A computer system on which at least a database management system is installed configured to be able to run a computer program implementing the method according to claim 1 to claim 6 or to be able to assist in performing the method according to claim 7.
9. An electronic data carrier storing data usable to install a computer program implementing the method according to claim 1 or to assist in performing the method according to claim 7.