Synthesizing Structured Text from Logical Database Subsets

Alkis Simitsis
IBM Almaden Research Center
San Jose, California, USA
asimits@us.ibm.com

Georgia Koutrika
Stanford University
Palo Alto, California, USA
koutrika@stanford.edu

Yannis Alexandrakis
National Technical University of Athens
Athens, Hellas
i.alex@dblab.ntua.gr

Yannis Ioannidis
University of Athens
Athens, Hellas
yannis@di.uoa.gr

ABSTRACT

In the classical database world, information access has been based on a paradigm that involves structured, schema-aware, queries and tabular answers. In the current environment, however, where information prevails in most activities of society, serving people, applications, and devices in dramatically increasing numbers, this paradigm has proved to be very limited. On the query side, much work has been done on moving towards keyword queries over structured data. In our previous work, we have touched the other side as well, and have proposed a paradigm that generates entire databases in response to keyword queries. In this paper, we continue in the same direction and propose synthesizing textual answers in response to queries of any kind over structured data. In particular, we study the transformation of a dynamically-generated logical database subset into a narrative through a customizable, extensible, and template-based process. In doing so, we exploit the structured nature of database schemas and describe three generic translation modules for different formations in the schema, called unary, split, and join modules. We have implemented the proposed translation procedure into our own database front end and have performed several experiments evaluating the textual answers generated as several features and parameters of the system are varied. We have also conducted a set of experiments measuring the effectiveness of such answers on users. The overall results are very encouraging and indicate the promise that our approach has for several applications.

1. INTRODUCTION

In the classical database world, information access has been based on a paradigm that involves structured, schema-aware, queries and tabular answers. In the current computing environment, however, where information prevails in most activities of society, serving people, applications, and devices in dramatically increasing numbers, this paradigm has proved to be very limited. On the query side, much work has been done on moving towards keyword queries (over structured data), which are much easier to handle for most users. On the answer side, however, there has been very little work to move to different paradigms. Indeed, a table may be ideal for many kinds of information, e.g., the daily transactions in a banking environment, a person's contacts list, or the movies shown in theaters around town, and for particular types of users, e.g., people with technical background. Nevertheless, there are many other scenarios, involving different users, applications, or query contexts, where other forms of answers are suitable. For example, enterprises often need subsets of their regular, large, databases that conform to the original schemas and satisfy all constraints for use in realistic tests of new applications before deploying them to production. In our previous work, we have dealt with the above scenario and have proposed the generation of entire databases in response to keyword queries [15, 22]. In this paper, we continue in the same direction and propose synthesizing *textual answers in response to queries* (of any form) *over structured data*.

As a motivating example from an actual, real-world situation, consider an international group of leading organizations that operate in the environmental sector, including ESA (European Space Agency), IMO (International Maritime Organization), UNESCO IOC (Intergovernmental Ocean Committee), several coast-guard offices, and others [1]. One typical activity in which these organizations are involved is the preparation of periodical reports on specific environmental topics of concern. For example, during the first few hours and days following an oil spill, scientists from these organizations retrieve data from multiple data sources (some archival, others injected with real-time data from satelite measurements) and continuously generate new assessment reports on the status of the marine and coastal environment in the area of the spill and the potential impact on its biological ecosystem. These reports are addressed to governmental agencies, they typically follow a prespecified layout, and their contents are essentially derived from the results of particular queries. By generating these results in textual form (and attaching any related raw data and graphs necessary), the reports could be generated almost entirely automatically, saving valuable time for the fight against the spill.

In the same spirit as above, textual answers can be useful in several other practical cases: a short description of a museum's exhibits, possibly customized to a visitor's particular interests; a brief history of a patient's medical conditions; the highlights of a collection in a digital library, with a few sentences on the main authors in the collection; a summary of a theater play in an information portal; and others. In such situations, textual answers are usually preferred by all users, independent of their technical experience. Moreover, the formation of textual answers becomes critical in all situations for people with visual impairments or reading disabilities. Using a speech recognizer [6, 20] to convert a speech signal to a query and a text-to-speech system (TTS) [4, 5, 20] to convert the textual form of the query answer into speech, these people would be given the chance to interact with information systems, orally pose queries, and listen to their answers.

In this paper, we study the transformation of the output of a query over a relational database, which we call a logical database subset, into a narrative. In our approach, which could be generalized to other data models as well, we exploit the structured nature of the underlying database schema and generate readable and concise query answers. In doing so, we face two main technical issues, which we address as follows: to enrich the semantics of the database schema graph, we resort to template logic and provide a language and a mechanism that allow the definition of customizable, reusable, and extensible templates for the automated translation of individual structures of the graph; to traverse the graph, we devise three generic translation modules, for three different schema formations, and offer a translation algorithm that synthesizes a narrative from the tuples contained in a logical subset.

Contributions. Specifically, the main contributions of this paper are the following:

- We introduce a framework for transforming structured data into structured text (Section 3) and provide a template mechanism and language for semantically annotating a database schema (Section 4).
- We formulate an algorithm for translating a logical database subset into a narrative that synthesizes the contents of the subset following a set of rules and templates. This includes three methods for translating arbitrary combinations of relations (modules) of three particular forms, respectively (Section 5): sequences of relations (unary modules); individual relations connected with two or more relations (split modules); and two or more relations connected to individual relations (join modules).
- We present our implemented prototype system and our experimental evaluation. In particular, we demonstrate how the narrative produced is affected as several features and parameters of our approach are varied, as well as how the response time of the system is affected by its configuration. We have also conducted a set of experiments measuring the impact that different answers, generated by different configurations, have on users. Overall, the results of our experiments are very encouraging and indicate a strong potential for our approach. (Section 6.)

2. RELATED WORK

The problem of facilitating information access to databases has two facets: facilitating querying and improving system answers.

In the first direction, keyword queries have been proposed as a way to relax the schema-aware, structured query paradigm over structured data (e.g., [2, 12, 14]) or semi-structured data (e.g., [8, 9, 10, 13]). Précis queries are unstructured queries over relational data with multi-relation databases as answers [15]. RDBMS vendors also present solutions that create full text indexes on text attributes of relations in order to make them searchable [7, 11, 16].

The problem of facilitating the non-technical user has been also discussed in the field of natural language processing (NLP). Several works are presented concerning NL Querying [17, 27], NL and Schema Design [18, 25], NL and DB interfaces [3, 19], and Question Answering [23, 26]. As far as we are aware of, related literature on NL and databases, has focused on totally different issues such as the interpretation of users' phrasal questions to a database language, e.g., SQL, or to the automatic database design, e.g., with the usage of ontologies [24]. Several recent efforts use phrasal patterns or question templates to facilitate the answering procedure [19, 23].

In earlier work, we have described the idea of constructing a précis , i.e., a close to natural language representation of information in a database, using the information conveyed by the database graph, which is properly annotated in order to enhance its semantics [21]. We have presented the translation of a single relation into stylized textual form that resembles a narrative and have provided a sketch for the translation of a simple join relationship between two relations. In this paper, we build upon the ideas suggested in [21] and significantly extend them as follows: (a) we provide a framework for the presentation of structured data as structured text; (b) we formally describe the translation of a logical subset; (c) we present a method for the translation of any arbitrary combination of relations schematically formed either as a unary or a split or a join module; (d) we support the translation of an arbitrary logical subset; (e) we identify the limitations of the translation and provide solutions for its applicability; (f) we present our implemented prototype system; and (g) we evaluate our approach both from an efficiency as well as an effectiveness viewpoint.

The translation process presented resembles those used for handling natural language querying over relational databases (e.g., [19, 23]) in that they all involve some amount of additional predefinitions for the meanings represented by relations, attributes and primary-to-foreign key joins. However, natural language query processing is more complex, since it has to handle ambiguities in natural language syntax and semantics whereas our approach uses well defined templates to rephrase relations and tuples. Furthermore, it has the advantage that it is not limited to any dictionary, because it concerns relational databases where the schemata are predictable and familiar to an expert, e.g., the dba; thus the template mechanism, presented later in this paper, is sufficient for this aim. Moreover, this work considers arbitrary, dynamically generated, logical database subsets whose form is only limited to the database schema graph. Works, such as [23], use a set of predefined question patterns, which cannot claim for completeness, i.e., this set is difficult to capture any possible query over a given database, and they produce pre-specified answers, where only the values in the patterns change. This is in contrast to our approach, which takes as input a logical subset and uses templates and constructs of sentences defined on the constructs of the database graph to generate dynamic answers. Template multi-utilization is a plus.

3. FRAMEWORK

In this section, we first introduce our data model and several key concepts that are pertinent to the problem we try to solve, and then, we formulate the translation problem.

Data Model. We consider a relational database D represented by its *database schema graph* $\mathbf{G}(\mathbf{V},\mathbf{E})$, which is a directed graph corresponding to the schema of D. There are two types of nodes in $\mathbf{V}$: (a) relation nodes, $\mathbf{R}$ - one for each relation in the schema; and (b) attribute nodes, $\mathbf{A}$ - one for each attribute of each relation in the schema. Likewise, edges in $\mathbf{E}$ are: (a) projection edges, $\mathbf{\Pi}$ - one for each attribute node, emanating from its container relation node and ending at the attribute node, representing the possible projection of the attribute in a query answer; and (b) join edges, $\mathbf{J}$ - emanating from a relation node and ending at another relation node, representing a potential join through a primary key - foreign key relationship between these relations. Therefore, a database schema graph is a directed graph $\mathbf{G}(\mathbf{V},\mathbf{E})$, where $\mathbf{V}=\mathbf{R}\cup\mathbf{A}$ and $\mathbf{E}=\mathbf{\Pi}\cup\mathbf{J}$. Since projection edges are always from relation nodes to attribute nodes, they are typically indicated without their direction, which is easily inferred by the types of the nodes. Both nodes and edges are annotated by appropriate template labels. (We will elaborate on these templates in section 4.) Since a join edge involves two relations, it may have two different directions; in our setting, we consider that an appropriate template label corresponds to each direction.

A *logical database subset* L of a database D is a subgraph of $\mathbf{G}(\mathbf{V},\mathbf{E})$ having the following properties: (a) it contains a subset of the relations in the original database schema, (b) it involves some of the attributes of each of these relations, (c) it contains a subset of the tuples that each of these relations had in the original database (projected on the set of attributes that are present in the subset), and (d) it contains a subset of the edges, along with their directions, in the original database graph. The nodes and edges of this subgraph are annotated by the respective template labels, which have been specified a priori in the database schema graph.

A logical database subset can be derived from a database in various ways. For example, it may be generated by a set of queries that aim at building a new, smaller, database that captures the relationships between tuples in the original one. Or it may be the output of a précis query [15, 22], which is an unstructured query whose answer is a multi-relation database that contains tuples that match the query terms as well as tuples that are implicitly related to them in the database aiming at providing greater insight to the user. How a logical subset is described and shaped, i.e., which relations, attributes, and tuples should be present in it or which join edges connect the relations involved and in which direction, and how it is generated are both issues that are orthogonal to our translation framework and algorithms. For instance, if it is the result of a précis query, its shape and tuples are determined by semantics and constraints of that query.

In order to synthesize a narrative from a given logical database subset L of D, the starting point (central element) of the narration needs to be given too. This point is a relation R_L, which will be referred to as the *initial relation* in the rest of the paper. Hence, the logical subset schema graph is a rooted graph with relation R_L as the root.

Example. *We consider a movie database described by the following schema; primary keys are underlined.*

$THEATRE(\underline{tid}, name, phone, region)$ $PLAY(\underline{tid}, \underline{mid}, \underline{date})$
$MOVIE(\underline{mid}, title, year, did)$ $GENRE(\underline{mid}, genre)$
$ACTOR(\underline{aid}, aname, blocation, bdate)$ $CAST(\underline{mid}, \underline{aid}, role)$
$DIRECTOR(\underline{did}, dname, blocation, bdate, nominat)$

An example logical subset containing information about the director Woody Allen, such as personal information and some movies that he has directed, is depicted in Figure 1. It contains two relations with a subset of their attributes and four tuples, one in relation DIRECTOR and three in relation MOVIE. The relation DIRECTOR (depicted in grey) is the initial relation. Directed edges show the direction of translation.

Heading attributes. Given a database D, in order to describe the semantics of a relation R_i along with its attributes in natural language, we consider that R_i has a *conceptual* and a *physical* meaning. Its conceptual meaning is expressed by an appropriate comment attached to the relation. For instance, in most modern DBMS's we can use the *COMMENT* statement to add a comment on a relation or an attribute. For simplicity in the presentation, we refer to the conceptual meaning of a relation (or an attribute) as its name. The physical meaning of a relation is represented by the name of one of its attributes that most characterizes the tuples of this relation. This attribute is termed the *heading* attribute and it is depicted as a hachured rounded rectangle.

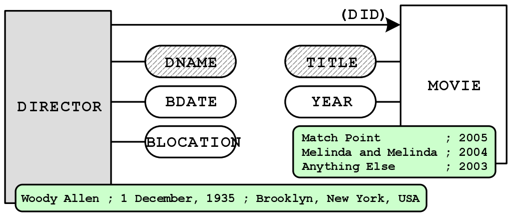


Figure 1: An example logical database subset

The heading attribute h_R of a relation R is defined as the attribute whose name represents the physical meaning of that relation. This attribute should always be present in the logical subset, which is to be translated, independently of the method or the requirements/constraints used for its creation. A domain expert makes the selection of heading attributes at the phase of initial construction of the database schema graph.

We do not anticipate that all relations should have a heading attribute. For instance, a relation used only for storing *n-to-m* relationships between different entities does not require a heading attribute. Usually, such relations are used only for interconnecting tuples found in other relations and have no attributes in the logical database subset (apart from those used in the join edges participating in the primary and foreign keys.)

Labels. Each node n is annotated by a label $l(n)$ that represents the conceptual meaning –i.e., the name– of the node and is stored as a comment on the specified node. For example, the label of the attribute node $TITLE$ is "title". Each projection edge $e\in\mathbf{\Pi}$ that connects an attribute A^i_j with its container relation R_i is annotated by a label that signifies the meaning, in terms of natural language, of the relationship between this attribute and the heading attribute of the respective relation. If a projection edge connects a relation node with its heading attribute, then the respective label reflects the relationship of this attribute with the conceptual meaning of the relation; e.g., "the $TITLE$ of a $MOVIE$". Each join edge $e\in\mathbf{J}$ between two relations has a label that signifies the relationship between the heading attributes of the relations involved; e.g., "the $GENRE$ ($.GENRE$) of a $MOVIE$ ($.TITLE$)". The label of a join edge involving a relation without a heading attribute signifies the relationship between its previous and subsequent relations. The labels are defined by the designer/administrator of the database. In the following section, we present a template mechanism that facilitates this procedure.

Example. *Consider the logical subset depicted in Figure 1. Relation MOVIE conceptually represents "movies" in real world; thus, a comment containing the concept MOVIE may express its conceptual meaning. Moreover, "title" is the main characteristic of a "movie"; thus, relation MOVIE should have TITLE as its heading attribute, since the "title" captures the physical meaning of a "movie". A possible label attached to the projection edge between relation MOVIE and its attribute YEAR may be: "the YEAR of a MOVIE (.TITLE)".*

Constraints. The translation of the information stored in one or more relations can be realized by different means. Therefore, for the translation, a set of constraints may be provided to facilitate the construction of answers following different formats. Such constraints may determine: (a) the length of a phrase created w.r.t. the number of words or lines contained in it; (b) the maximum number of tuples that should be considered in the result; and (c) the means by which different design structures should be translated. We will elaborate on the functionality and the usage of the constraints in the presentation of our translation algorithm (see subsection 5.3.)

Problem Description. Given the schema graph **G** describing a database D, a logical subset L of D, an initial relation R_L in it, and a set of translation constraints, the problem of synthesizing text from a logical database subset is defined as constructing a textual synthesis of the tuples in the subset that captures the semantics conveyed by the database schema $\mathbf{G}(\mathbf{V},\mathbf{E})$ and the correlation of relations through primary and foreign keys starting from the initial relation.

4. TEMPLATE MECHANISM

The text synthesis of tuples contained in a logical subset follows the database schema and the correlation of relations through primary and foreign keys. This process is facilitated by alphanumeric expressions called *template labels*. First, we define the template labels, and then, we present a language for their construction.

Templates. A *template label*, $label(u,z)$, is assigned to each edge $e(u,z) \in \mathbf{E}$ of the database schema graph $\mathbf{G}(\mathbf{V},\mathbf{E})$. This label is used for the interpretation of the relationship between the values of nodes u and z in a narrative. The template label $label(u,z)$ of an edge $e(u,z)$ is an alphanumeric expression constructed by the concatenation of the following elements: (a) the label, $l(u)$ –i.e., the name– of the starting or source node; (b) the label, $l(z)$ –i.e., the name– of the ending or target node; (c) a finite set of alphanumeric expressions, $expr_i$; and (d) a finite set of constructs supported by our template language presented next. A simple template label may have the form:

$$label(u,z) = expr_1 + l(u) + expr_2 + l(z) + expr_3 \qquad (1)$$

where $expr1$, $expr2$, $expr3$ are alphanumeric expressions and the operator "+" acts as a concatenation operator.

A template label for a node contains the label of the node and a finite set of expressions. However, for simplicity of the presentation, we will consider that such a template is the label of the node.

In order to use template labels or to register new ones, we use a simple language for templates that supports variables, loops, functions, and macros.

In a template, when we refer to the conceptual meaning of a node, we simply use its name. When an instance of the node is needed, then we use the node as a variable. There are two kinds of variables: *parameter variables* and *loop iterators*.

Parameter Variables. Parameter variables are marked with a @ symbol at their beginning and they are replaced by values at instantiation time. In several cases, the values returned in a query result from a certain attribute could be more than one. Then, we use a list of parameters denoted as:

$$@ < parametername > [\,]$$

For such lists, their length is provided at instantiation time.

Example. *A template label for the projection edge $e(THEATRE, PHONE)$ could be:*

$label(THEATRE, PHONE) =$
"The " + $l(PHONE)$ + " of the " + $l(THEATRE)$ +
$@THEATRE.NAME$ + " is " + $@THEATRE.PHONE$

where $l(PHONE)$ = "phone" and $l(THEATRE)$ = "theatre" stand for the conceptual meaning of the attribute PHONE and the relation $THEATRE$, respectively. Assuming that the parameter values $@THEATRE.NAME$ and $@THEATRE.PHONE$ get the values "ALPHAVILLE" and "12345", a valid narrative for this edge is:

"The phone of the theatre ALPHAVILLE is 12345"

Loop iterators. Loop iterators are implicitly defined in the loop constraint, as we will discuss later. In each round of the loop, all the properly marked appearances of the iterator in the loop body are replaced by its current value (similarly to the way a C preprocessor treats $\sharp DEFINE$ statements). Iterators that appear marked in the loop body are instantiated even when they are part of another string or a variable name. We mark such occurrences by enclosing them between $'s. This functionality enables referencing all values of a parameter list and facilitates the creation of an arbitrary number of pre-formatted strings.

Functions. We employ a built-in function:

$$arityOf(< list\_of\_parameters >)$$

which returns the arity of a list of parameters, mainly in order to define upper bounds in loop iterators. All the well-known string manipulation functions are supported too.

Loops. Loops enhance the genericity of the templates by allowing the designer to handle templates with unknown number of variables and with unknown arity for parameters involved. The general form of loops is:

$$[< simple\ constraint >]\ \{(loop\ body)\} \qquad (2)$$

where simple constraint has the form:

$$<lower><operator><iterator><operator><upper>$$

We consider only linear increase with step equal to 1. Upper bound and lower bound (default value 1) can be arithmetic expressions involving $arityOf()$ function calls, variables and constants. Valid arithmetic operators are $+$, $-$, $/$, $*$ and valid comparison operators are $<$, $>$, $=$, all with their usual semantics. During iterations, the loop body is reproduced and at the same time all marked appearances of the loop iterator are replaced by its current value, as described before. Loop nesting is permitted.

Example. *Consider the following case:*

$$[i \leq arityOf(MOVIE)]\ \{MOVIE\_\$i\$\}$$

The lower bound has the default value 1*, and the upper bound is limited to the number (arity) of attributes of the relation MOVIE. Thus, the iterator i takes values between* 1 *and the total number of attributes of MOVIE. The loop body contains a parameter list that stores the attributes involved in the relation MOVIE. After the evaluation of the function arityOf, the loop that represents the attributes of MOVIE has the following form:*

$$[i \leq 2]\ \{MOVIE\_\$i\$\}$$

and at the instantiation of the parameters, the following results are obtained: MOVIE_1 = TITLE (first attribute) and MOVIE_2 = YEAR (second attribute).

Macros. We introduce macros to ease the definition and to improve the readability of templates. Macros facilitate attribute and variable name expansion. For instance, one major problem in defining a language for templates is the difficulty in dealing with attributes or attribute values of arbitrary arity. At the template level, it is not possible to pin-down the number of (a) attributes that are projected in the logical subset, and (b) values of the involved attributes, to a specific value. Hence, in order to find out:

type	template macro	instantiated macro	evaluated macro
series of attributes	$DEFINE\ MOVIES\_LIST$ *as* $[i < arityOf(MOVIE)]\ \{MOVIE\_\$i\$,\}$ $[i = arityOf(MOVIE)]\ \{MOVIE\_\$i\$\}$	$DEFINE\ MOVIES\_LIST$ *as* $[i < 2]\ \{MOVIE\_1,\}$ $[i = 2]\ \{MOVIE\_2\}$	$MOVIES\_LIST = \{$ $TITLE,$ $YEAR\ \}$
series of values	$DEFINE\ MOVIES\_TITLES\_LIST$ *as* $[i < arityOf(@MOVIE.TITLE)]$ $\{@MOVIE.TITLE[\$i\$],\}$ $[i = arityOf(@MOVIE.TITLE)]$ $\{@MOVIE.TITLE[\$i\$]\}$	$DEFINE\ MOVIES\_TITLES\_LIST$ *as* $[i < 3]$ $\{@MOVIE.TITLE\_1,\}$ $\{@MOVIE.TITLE\_2,\}$ $[i = 3]$ $\{@MOVIE.TITLE\_3\}$	$MOVIES\_TITLES\_LIST = \{$ “Match Point”, “Melinda and Melinda”, “Anything Else” }

Figure 2: An example macro instantiation concerning the relation $MOVIES$ depicted in Figure 1

(a) the attributes of a relation R projected in a certain logical subset, we create a series of attributes as follows:

$DEFINE\ REL\_R\_LIST$ *as*
$[i < arityOf(R)]\ \{R\_\$i\$,\}$
$[i = arityOf(R)]\ \{R\_\$i\$\}$

(b) the values of an attribute A of a relation R corresponding to a certain logical subset, we create a series of values as follows:

$DEFINE\ REL\_R\_ATTR\_A\_LIST$ *as*
$[i < arityOf(@R.A)]\ \{@R.A[\$i\$],\}$
$[i = arityOf(@R.A)]\ \{@R.A[\$i\$]\}$

Note that the existence of the two loops in each macro serves to avoid the presence of an erroneous “,” after the last value in a list.

Example. *For the example logical subset of Figure 1, the determination of the attribute and value series for the relation MOVIE and its attribute TITLE, respectively, is realized as shown in Figure 2 (from left to right.)*

5. TRANSLATION

In this section, we describe a method that parses a logical database subset and composes a synthesis of tuples in a narrative. The nodes and edges of the logical subset are already annotated by the appropriate template labels, which have been constructed by an administrator following the mechanism presented in the previous section.

The translation starts from the initial relation and continues following the correlation of the relations through the edges of the logical subset. Each clause created renders information stored in a relation; e.g., “Woody Allen was born in New York”, and/or captures the relationships between tuples stored in relations interconnected through join edges; e.g., “Woody Allen has directed the movie The Jade Scorpion”. These relations will be referred as *neighbors*. The translation terminates when the traversal of the logical subset graph is complete. At this point, all clauses generated are listed.

In what follows, we first describe the translation of the information stored in a single relation. Then, we present the construction of phrases containing information stored in multiple relations, by using three generic modules that capture the possible schematic representations that can be met in the traversal of the logical subset graph. Finally, we introduce an algorithm that combines the above for the translation of a logical subset.

5.1 Translation of a single relation

The translation of a single relation results in sentences constructed by using the values stored in it. For each tuple of this relation a different sentence is created. The coalescence of all sentences is driven by the template assigned to the relation by the designer.

There are two alternatives for the representation of a relation’s content:

(a) using only the heading attribute of the relation or

(b) constructing a phrase determined by one or more macros that combines the information stored in the heading attribute of a relation and in the rest of attributes projected in the logical subset.

The translation of information stored in a relation R is annotated as R°, when only the heading attribute of R is considered (case a), and as $R^\bullet$, when information from all the attributes of R is considered (case b).

In the first case, if the translation mode chosen is R°, then the clause produced contains only the heading attribute and the template label of its projection edge. Usually, such clauses are subordinate clauses, thus, they cannot form separate sentences but they can form a sentence when joined with a main clause; this is realized in the translation of multiple relations (see 5.2.)

In the second case, if the translation mode chosen is $R^\bullet$, exhaustively, a different clause per attribute should be considered. Since the value of the heading attribute characterizes the information stored in a relation, it is used as the subject and comprises the first part of each of the sentences. The labels of the projection edges found in the logical subset are evaluated next. For multiple attributes of the same relation, inevitably, the same subject has to be repeated many times. To avoid this, a domain expert should attach suitable expressions to the projection edges to allow the construction of composite meaningful sentences. This is facilitated by a simple find-and-replace mechanism, *resolve_common_expressions*, which finds common expressions in the clauses that correspond to the label attached to each projection edge.

Example. *For the translation of relation DIRECTOR depicted in Figure 1 in the mode $R^\bullet$, assume that the labels of the projection edges that connect the relation –practically, its heading attribute, DNAME– to its attributes BDATE and BLOCATION, which store information about the birth date and birth location of a director, are the following:*

$$label(DIRECTOR, BDATE) = \\ @DNAME\ +\ \text{“ was born”}\ +\ \text{“ on ”}\ +\ @BDATE$$

$$label(DIRECTOR, BLOCATION) = \\ @DNAME\ +\ \text{“ was born”}\ +\ \text{“ in ”}\ +\ @BLOCATION$$

When both attributes are involved in the answer, the application of the resolve_common_expressions mechanism identifies that @DNAME and “ was born” are common expressions. Finally, the clause derived from the DIRECTOR relation can be as follows:

```
Algorithm Translation of a Relation ($TR$)
Input:   a relation $R$, a logical subset $L$, constraints $T$,
         translation mode ($R^{\bullet/\circ}$)
Output:  an array of sentences $Sentences[\,]$
Begin
1.  Let $h_R$ be the heading attribute of $R$
2.  If $R^\circ$ mode {
    2.1 Foreach tuple $t \in R$, $R \in L$, w.r.t. the constraints $T$ {
        2.1.1 $Sentences[t] = l(h_R)$
    }}
    Else {
    2.2 Foreach tuple $t \in R$, $R \in L$, w.r.t. the constraints $T$ {
        2.2.1 empty $Sentences[t]$ and $clause[\,]$
        2.2.2 $clause[t, h_R] = l(h_R)$
        2.2.3 Foreach attribute $A$ in $R$, $A \neq h_R$ {
                  $clause[t, A] = label(h_R, A)$
              }
        2.2.4 $Sentences[t] \leftarrow resolve\_common\_expressions(clause[\,])$
    }}
3.  Return $Sentences[\,]$
End
```

Figure 3: Algorithm *TR*

"@$DNAME$ was born on @$BDATE$ in @$BLOCATION$"

Formally, the procedure for the translation of the information stored in a relation is described by the algorithm *TR* (Figure 3).

5.2 Translation of multiple relations

A logical subset of a database usually contains more than one relation, and constitutes a graph. As the translation procedure parses this graph, following the correlation of relations through primary and foreign keys, three design patterns can be identified, as shown in Figure 4. The first one, M_U, represents the simple connection of two subsequent relations though a join edge (unary module). The second one, M_S, represents the connection of a source relation to more than one target relation (split module). The third one, M_J, represents the connection of more than one source relation to a single target relation (join module).

In order to describe the translation of multiple relations, we first describe how this procedure is realized in each one of these modules. In the next subsection, we present an algorithm that describes the translation of the whole graph.

Unary module. Let us first examine the simple case of two subsequent (neighboring) relations R_i and R_j, depicted in Figure 4 as the unary module, M_U. Essentially, if we consider the unary module in abstraction, then we have to translate the information residing in both relations R_i and R_j, taking into consideration the relationship that joins the two relations. This translation consists of two parts: one part representing the information stored in the source relation R_i of the join edge and one part corresponding to the target relation R_j and following the join edge between the two relations. Each of these parts may comprise more than one sentence.

The first part is constructed as discussed in the previous subsection. We elaborate here on the construction of the sentences of the second part. In order to capture the relationship between the two relations, the subject of the respective sentence is the heading attribute of the source relation R_i in the join sequence. The rest of the sentence comprises an expression that usually contains a verb phrase and a clause representing the information stored in the target relation R_j. The verb phrase is stored as an expression in the template label tagged on the join edge that connects the relations. There are two ways to represent the information stored in relation R_j in this template:

- using only its heading attribute (R_j° mode) or

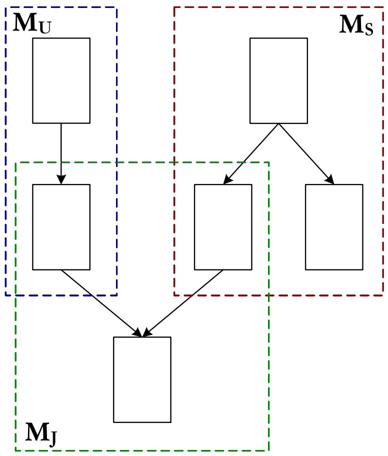


Figure 4: Different cases for translation of multiple relations

- constructing a phrase combining the information stored in all its attributes ($R_j^\bullet$ mode).

In the former case, in order to complete the translation of the target relation, an extra clause is needed representing the information contained in the remaining attributes of the target relation R_j. This sentence is constructed as described in subsection 5.1 considering that the target relation is translated in $R_j^\bullet$ mode.

The aforementioned analysis can be formally summarized as follows. For the unary module M_U comprising two relations R_i and R_j with a direction from the former to the latter, the translation is realized in two steps:

(i) evaluation of $R_i^\bullet$ (as described in subsection 5.1)

(ii) evaluation either of

$$(R_i^\circ - R_j^\circ) \uplus R_j^\bullet \quad \text{or}$$
$$(R_i^\circ - R_j^\bullet)$$

where $R_i - R_j$ stands for the creation of a phrase having as parts the information adapted from R_i and R_j, represented by an appropriate template attached to the join edge between R_i and R_j. The symbol $\uplus$ represents the concatenation of different sentences. At the end, the sentences produced by the two steps are combined to produce the final result.

Consequently, the translation of a unary module first generates a sentence containing the complete information stored in the source relation (step (i)). Then, there are two alternatives to follow in step (ii). Following the first one, a set of phrases is constructed consisting of: (a) a simple phrase of the form "subject - verb - object" with the possible addition of some fixed expressions to complement the phrase, and (b) a phrase for the complete translation of the second relation of the form "subject - verb - several subordinate clauses".

Hence, we observe in step (ii) that the two alternatives are ordered by increasing order of construction's complexity. The second alternative constructs a more compact, thus more complex, and appealing phrase. However, the production of such a sentence is not always feasible due to the potential constraints imposed; in such case, the first alternative should be followed. In general, the choice of one of them is driven by the constraints given, but this is further explained in the subsection 5.3.

Example. *In the example of Figure 1, the relation DIRECTOR is connected to MOVIE through the DID key. This case matches with the unary module. The subject of the respective clause for the translation of the single relation DIRECTOR will be the DNAME* *attribute. We construct the template clause that corresponds to the DIRECTOR relation (step (i)) as described in subsection 5.1:*

$$@DNAME + \text{" was born"} + \text{" on "} + @BDATE + \text{" in "} + @BLOCATION$$

Next, we should choose any of the two alternatives of step (ii) w.r.t. the constraints given. Let us consider that the constraints allow the choice of the second one: $R_i^\circ - R_j^\bullet$. Assume, that the respective template clause for the MOVIE relation is the following:

$$@TITLE + \text{" ("} + @YEAR + \text{")"}$$

Since the relation MOVIE may contain more than one tuple, a macro is needed to iterate the above template among the tuples. We proceed with the clause composed by the join relationship that connects DIRECTOR and MOVIE. The template label of this relationship can be represented by the following formula:

$$label(DIRECTOR, MOVIE) = expr_1 + @DNAME + expr_2 + MOVIE\_LIST$$

The macro MOVIE_LIST and expressions may be defined as:

```
DEFINE MOVIE_LIST as
    $[i < arityOf(@TITLE)]$
        $\{@TITLE[\$i\$] + \text{" ("} + @YEAR[\$i\$] + \text{"),"}\}$
    $[i = arityOf(@TITLE)]$
        $\{@TITLE[\$i\$] + \text{" ("} + @YEAR[\$i\$] + \text{")."}\}$
$expr_1 \leftarrow$ "As a director, "
$expr_2 \leftarrow$ "'s work includes "
```

Therefore, after the instantiation of the template (see Figure 2) and the appropriate concatenations, the result for "Woody Allen" located in the relation DIRECTOR is:

"Woody Allen was born on December 1, 1935 in Brooklyn, New York, USA. As a director, Woody Allen's work includes Match Point (2005), Melinda and Melinda (2004), Anything Else (2003)."

If we had considered the first alternative of step (ii), then the result would be different in that the translation of the information residing in relation MOVIE would be represented by more than one clause. A possible interpretation of this w.r.t. the example of Figure 1 might be the following:

"Woody Allen was born on December 1, 1935 in Brooklyn, New York, USA. As a director, Woody Allen's work includes Match Point, Melinda and Melinda, Anything Else. Match Point was released in 2005. Melinda and Melinda was released in 2004. Anything Else was released in 2003."

With the resolve_common_expressions mechanism mentioned in the previous subsection, a more elegant result would be produced.

However, the two results have some critical differences. The first one is more compact, it does not have any overlaps, and it resembles natural language in a great extent. On the other hand, its creation is more complex and in some cases even infeasible. For instance, if the relation MOVIE in the logical subset contained more attributes and the creation of more than one clause was needed to describe them, then such elegant result is difficult to be created. The second result is constructed in a straightforward manner and it consists of a coalescence of several simple phrases. This kind of synthesis can describe more complex logical subsets.

Split module. Another case to be examined concerns the correlation between a single source relation and more than one target relation, depicted in Figure 4 as the split module M_S. As before, the description of the module comprises two parts: one that corresponds to the source relation and one that describes the connection between the source and target relations involved. The first part is constructed as described in subsection 5.1.

The second part contains a set of sentences capturing the join relationships among relations involved along with the information stored in the target relations. A sentence created using information spread in different relations has as a subject the heading attribute of the source relation. The rest of the sentence comprises a verb phrase and one or more clauses representing the information stored in the target relations. Although it is possible to consider the case as two separate unary modules, there is the problem of repeating information from the source relation. To avoid this issue, the information adapted from the neighboring relations is presented as one phrase combined with a coordinating conjunction (joining word) θ; the default word in our approach is the "and". The components of this combined phrase can be either in R° or $R^\bullet$ mode.

Thus, formally, for the split module M_S comprising relations R_i and R_{j_1}, R_{j_2}, with direction from the former to the latter, the translation is realized in two steps:

(i) evaluation of $R_i^\bullet$ (as described in subsection 5.1)

(ii) evaluation either of

$$R_i^\circ - (R_{j_1}^\circ\ \theta\ R_{j_2}^\circ) \uplus R_{j_1}^\bullet \uplus R_{j_2}^\bullet \quad \text{or}$$
$$R_i^\circ - (R_{j_1}^\bullet\ \theta\ R_{j_2}^\circ) \uplus R_{j_2}^\bullet \quad \text{or}$$
$$R_i^\circ - (R_{j_1}^\bullet\ \theta\ R_{j_2}^\bullet)$$

Observe that the possible choices in step (ii) are listed by increasing order of construction's complexity in the same sense described above for the unary module. At the end, the results of both steps are combined to the final phrase. The above analysis is also valid for a split module comprised of more than two target relations: R_{j_1}, ..., R_{j_k}, with the respective combinations of $R_{j_x}^\bullet$ or $R_{j_x}^\circ$ mode.

Example. *Consider the hypothetical logical subset that comprises three relations $MOVIE(title, year)$, $DIRECTOR(name, birth\_location)$, and $ACTOR(name, nationality)$ (the $title$ and $name$ attributes are considered as the heading attributes.) Also, for simplicity in presentation, the keys are omitted. In each relation there is one tuple, let us say having for the heading attributes the values t_l^M, t_l^D, and t_l^A, respectively. Assume that the schema of this logical subset is as follows: $DIRECTOR \leftarrow \underline{MOVIE} \rightarrow ACTOR$. Then, this scenario resembles a split module, where R_i is MOVIE while R_{j_1} and R_{j_2} are the other two relations. Considering some exemplary expressions in the template labels, each one of the aforementioned alternatives for the step (ii) may produce the clauses depicted in the top part of Figure 5. Observe that the first alternative produces simpler clauses with some overlap, while the second and mainly the third alternative produce more complex but more elegant results. In all cases, the default word "and" is used as the joining word θ. Also, in the last two cases the template label used is slightly different as the respective expressions were enriched by the word "who". (A technicality in our implementation provides the functionality to store different template labels on the edges of the graph; one for each module.)*

Join module. The third case to be examined concerns the correlation between several source relations and a single target relation, depicted in Figure 4 as the join module M_J.

In this case, it is practically meaningless to combine the full descriptions of all relations. Thus, the information stored in the source relations that join to the target one, should have already been translated before the translation of the join module is realized. This

split module	$\underbrace{R_i^\circ - (R_{j_1}^\circ\ \theta\ R_{j_2}^\circ)} \uplus \underbrace{R_{j_1}^\bullet} \uplus \underbrace{R_{j_2}^\bullet}$	*The movie t_l^M involves the director t_l^D and the actor t_l^A. The director t_l^D was born in Italy. The actor t_l^A is Greek.*
	$\underbrace{R_i^\circ - (R_{j_1}^\bullet\ \theta\ R_{j_2}^\circ)} \uplus \underbrace{R_{j_2}^\bullet}$	*The movie t_l^M involves the director t_l^D who was born in Italy and the actor t_l^A. The actor t_l^A is Greek.*
	$\underbrace{R_i^\circ - (R_{j_1}^\bullet\ \theta\ R_{j_2}^\bullet)}$	*The movie t_l^M involves the director t_l^D who was born in Italy and the actor t_l^A who is Greek.*
join module	$\underbrace{(R_{i1}^\circ\ -\ R_j^\circ)} \uplus \underbrace{(R_{i2}^\circ\ -\ R_j^\circ)} \uplus \underbrace{R_j^\bullet}$	*The actor t_l^A is involved in the movie t_l^M. The director t_l^D is involved in the movie t_l^M. The movie t_l^M is released in* 2006.
	$\underbrace{(R_{i1}^\circ\ -\ R_j^\bullet)} \uplus \underbrace{(R_{i2}^\circ\ -\ R_j^\circ)}$	*The actor t_l^A is involved in the movie t_l^M which is released in* 2006. *The director t_l^D is involved in the movie t_l^M.*
	$\underbrace{(R_{i1}^\circ\ \theta\ R_{i2}^\circ)\ -\ R_j^\circ} \uplus \underbrace{R_j^\bullet}$	*The actor t_l^A and the director t_l^D are involved in the movie t_l^M. The movie t_l^M is released in* 2006.
	$\underbrace{(R_{i1}^\circ\ \theta\ R_{i2}^\circ)\ -\ R_j^\bullet}$	*The actor t_l^A and the director t_l^D are involved in the movie t_l^M which is released in* 2006.

Figure 5: Construction of example sentences using the different alternatives of the split and join modules

means that all source relations of the join module participate in its translation in R° mode. This observation guides the next definition.

Formally, for the join module M_J comprising relations R_{i1}, R_{i2} and R_j, with direction from the source R_{ix} relations to the target relation R_j, the translation is realized as follows:

(i) evaluation of $R_{i1}^\bullet$ (as described in subsection 5.1)

(ii) evaluation of $R_{i2}^\bullet$

(iii) evaluation either of

$$(R_{i1}^\circ - R_j^\circ) \uplus (R_{i2}^\circ - R_j^\circ) \uplus R_j^\bullet \text{ or}$$
$$(R_{i1}^\circ - R_j^\bullet) \uplus (R_{i2}^\circ - R_j^\circ)$$
$$(R_{i1}^\circ\ \theta\ R_{i2}^\circ) - R_j^\circ \uplus R_j^\bullet \text{ or}$$
$$(R_{i1}^\circ\ \theta\ R_{i2}^\circ) - R_j^\bullet \text{ or}$$

The candidate choices in step (iii) are listed by increasing order of construction's complexity in the same sense described before for the other modules. The final phrase is produced by the appropriate concatenation of the individual results of all three steps. The above analysis is also valid for a join module comprised of more than two source relation: R_{i1}, ..., R_{ik}, with the respective combinations of $R_{ix}^\bullet$ or R_{ix}° mode.

In the special case where the information stored in R_{i1} and R_{i2} refers to the same term, it is possible to avoid the second step and to simplify the expression $(R_{i1}^\circ\ \theta\ R_{i2}^\circ)$ with a process similar to the *resolve_common_expressions* of the algorithm TR (Figure 3).

However, there is also the option to translate several or all the parts of the join module as individual unary modules. This should be the typical case when: (a) the source relations contain information related to different terms; even if the terms are the same, extra knowledge that they refer to the same physical entity is needed; (b) the interpretation of the information stored produces large phrases w.r.t. the constraints given; or (c) there is a requirement to differentiate the information stored in different paths; e.g., one may want separate reports on the movies that Woody Allen has directed and has acted in, while another may want a full list of the movies that Woody Allen has participated in.

Example. *Consider the hypothetical logical subset presented in the split module but, in this case, with a different schema: $DIRECTOR \rightarrow \underline{MOVIE} \leftarrow ACTOR$. Then, this scenario resembles a join module, where R_j is $MOVIE$ and R_{i1} and R_{i2} are the other two relations. Considering some exemplary expressions in the template labels, each one of the aforementioned alternatives for step (iii) may produce the clauses depicted in the bottom part of Figure 5. Notice that as we pick a lower alternative, the result becomes more complex and appealing. On the other hand, when such merging of individual results is not feasible then a higher alternative should be chosen. In such case, the answer contains more and simpler clauses. A discussion on the decision of such an alternative is given in the following section.*

5.3 Generalization of results

Once we have described the building blocks needed, we elaborate on the translation of a logical subset L given an initial relation R_L and a set of constraints T. The result of this procedure is a synthesis of simple clauses. The translation is realized by the algorithm *Translation of a Logical Subset*, *TRLS*, which is presented in Figure 6 and described below. The algorithm takes as input a logical subset and a set of constraints, and it produces a textual answer.

$TRLS$ traverses the logical subset graph starting from the initial relation R_L of the respective initial subgraph. $TRLS$ constructs a different phrase for each tuple of R_L. The traversal continues to the neighboring relations following the directed edges of the graph, until the whole graph is explored.

```
Algorithm Translation of a Logical Subset (TRLS)
Input:    a logical subset $L$, an initial relation $R_L$, constraints T
Output:   an array of phrases $Text[\,]$
Begin
1.  $Text[L]$ = ' '
2.  Start from the root $R_L$
3.  Foreach tuple $t \in R_L$ {
      3.1 $QP \leftarrow \{R_L\}$
      3.2 While ($QP$ not empty) {
            a. pick head $R_i$ of $QP$
            b. If $in\text{-}deg(R_i) \leq 1$ {
                 $QP \leftarrow$ find the neighboring relations $R_{j_1}$, ..., $R_{j_k}$
                 If $out\text{-}deg(R_i) = 1$
                     { evaluate $M_U(R_i, R_j)$ }
                 If $out\text{-}deg(R_i) > 1$
                     { evaluate $M_S(R_i, R_{j_1}, ..., R_{j_k})$ }
               }
            c. Else
                 If all paths from sources $R_{j_1}$, ..., $R_{j_k}$ to $R_i$ are met
                     { evaluate $M_J(R_{j_1}, ..., R_{j_k}, R_i)$ }
                 Else
                     { continue with next $R_i$ }
            d. update ${R_i}^{\bullet/\circ}$ flag for $R_i$
            e. update $Text[S]$
      }
4.  Return $Text[\,]$
End
```

Figure 6: Algorithm *TRLS*

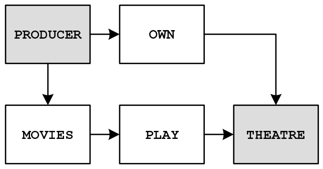


Figure 7: Example use of a join module

In this procedure, a list QP is used, which stores the neighboring relations of the relation currently processed. The way that these relations are placed in QP determines how each subgraph is explored.

Each time a relation R_i is visited, $TRLS$ matches it with one of the three patterns previously presented (subsection 5.2), according to the connection of this relation with its sources. If R_i has at most one source relation, it may participate in a unary module (when R_i's out-degree equals to 1) or a split module (when R_i's out-degree is greater than 1.) In this case, the target relation(-s) of the module are added in QP. When R_i has more than one source relation, then it participates in a join module as the target relation. Then, if all the paths from the source relations toward R_i have been considered, the algorithm interprets the information of the join module. Otherwise, it continues with another relation until all relations that participate in the join module are fully translated and then, the join module is ready to be processed. Given that there are no cycles contained in the logical subset, this policy does not drive to a deadlock [22].

Observe that the case of a join module does not signify the existence of a cycle. An example case is depicted in Figure 7. Starting from the initial relation $PRODUCER$ it is possible to reach relation $THEATRE$ via two ways: a producer has produced a movie that is played in a theatre and a producer owns a theatre.

Each relation R_i is annotated with a ${R_i}^{\bullet/\circ}$ flag corresponding to the $R_i^\bullet$ or R_i° mode. After the processing of the appropriate module, this flag is updated. When a relation has been fully translated, the respective flag for $R_i^\bullet$ mode is activated and from that time, the information stored in this relation may be evaluated only in R_i° mode. Hence, translating the same entity more than once is avoided and respectively, some steps in the evaluation of a module indicating the translation in $R_i^\bullet$ mode are omitted.

Each time a module is evaluated, the ambiguity in the choice of the proper expression in steps (ii) or (iii) (as presented in the formal definition of each module in subsection 5.2) of the translation of each module, is resolved as follows.

- The state of a relation's flag ${R_i}^{\bullet/\circ}$ should be taken into consideration.
- The constraints T should be satisfied; for instance, such constraints may determine (a) the length of a phrase w.r.t. the number of words or lines; (b) the maximum number of tuples that should be included in a (R° - $R'^\bullet$) phrase, before it should split in (R° - R'°) $\uplus$ $R'^\bullet$ phrase; and (c) the translation mode; e.g., the whole translation should be realized in R° mode.
- The choice of a candidate expression is driven by the goal of creating the most feasible compact answer w.r.t. the constraints given. For this reason the hierarchy that represents the increasing order of complexity in the answer's construction is used. Each time, the lowest feasible alternative in the hierarchy of the candidate choices in step (ii) (in unary and split module) or in step (iii) (in join module) is favored.

5.4 Applicability issues

The translation process is generic, in the sense that allows any policy to be chosen. One of the most prominent approaches is using a weighting scheme on the graph relations. In this way, the most significant results would be presented first. Going a step further, tuples in each relation could be ranked. That would allow for the most significant ones to be presented first and also for less significant tuples to be skipped using appropriate constraints. Additionally, it is possible to have different settings (e.g., different heading attributes and labels on nodes and edges) in order to produce customized narratives for multiple users or group of users. However, this issue does not affect the applicability of the approach presented, because the choice such settings is made before the translation takes place.

Although our approach technically works with databases of any size, translation of a very large database, in terms of relations, attributes, or tuples, will most likely not lead to meaningful or concise answers. The great usefulness of this approach is in describing small (subsets of) databases; hence, our focus is on query answers that constitute logical database subsets. Moreover, the intention of this work is not the construction of a human-intelligent system, this is not feasible. Thus, our method has limitations that are related to the length of the final output. In our understanding and experience gained by the interaction with users (Section 6), a meaningful and even a fascinating answer should be a short one. In most cases, it is practically useless to produce a narrative containing all information found in a database. Rather, it is preferable to produce short comprehensive answers. Therefore, the restraint on the result's length seems as a desideratum. There are several ways to achieve this; we mention the most prominent.

- To provide users with the capability to limit the containment of the answer by putting appropriate constraints, such as those we have already mentioned (Sections 3 and 5.3.)
- To consider translation only in ${R_i}^\circ$ mode; i.e., only phrases containing information derived from heading attributes. Then, each word/phrase of a heading attribute may be enriched with a *hyperlink* that guides to the full description of the respective

relation ($R_i^\bullet$ mode.) Another more conservative approach signifies the enrichment with hyperlinks only of the most "interesting" heading attributes.

- To split the results into paragraphs; e.g., for each tuple in the initial relation.

In the following section, we present our experimental results in order to evaluate the efficiency as well as the effectiveness of the logical subset translation.

6. EVALUATION

6.1 System Prototype

We have implemented a prototype system for the translation of logical subsets, which consists of three main modules: the translator engine, the metadata repository, and the graphical user interface (GUI). The translator engine is implemented in C++ and uses the LEDA 5.0 library to handle graphs. The engine implements the template language presented in Section 4 and provides the database graph traversal functionality described in Section 5. The metadata repository is stored in an Oracle 10g RDBMS and contains the metadata needed for the translation process; e.g., the template label for each link between a relation and another relation or an attribute. Personalized annotation of the database graph for different users or user groups is applicable through storage of different template profiles in the metadata repository. The GUI is implemented in Java and is to be used by the administrator of each application (usually only once) to tune the translation and annotate the database graph with appropriate template labels. To help the designer in this cumbersome task, the GUI provides the following features:

- *Easy navigation*. The database is representated as a graph, which can be manipulated in diverse ways: different representation levels, zoom-in/out, rotation, hyperbolic node distance correction, node expansion/collapse of a subgraph, etc. Figure 8(a) depicts a snapshot of the GUI with a movie database graph.

- *Easy supervision*. It is possible to preview a single relation or two relations interconnected through a join edge. The preview may show either a graph representation of the entity(-ies) involved, e.g., snapshot for relation *MOVIE* in Figure 8(c), or the template labels for the edges involved, e.g., join edge connecting relations *MOVIE* and *GENRE* in Figure 8(d). (Observe that the template label consists of the heading attribute of the first relation (*TITLE*), an expression, and a macro responsible for the translation of the second relation (*MGENRES_HA_LIST*).)

- *Easy annotation*. There is template editor that supports all the constructs of the template language introduced in Section 4 and automates the construction of macros. Figure 8(b) demonstrates a snapshot of that editor concerning a macro for the dynamic representation of the tuples stored in relation *MOVIES*. A typical macro construction for a specific relation can be done through simple 'drag-n-drop' operations as follows: after the relation and the desired attributes are chosen, the designer determines the macro type (tuple- or attribute-based), the translation parameters (e.g., the translation mode, $R^{o/x}$), formatting symbols (e.g., the line delimiter), and so on. Also, a macro may be stored in the repository and afterwards a new macro can be created from the stored one.

6.2 Experiments

We have performed several experiments to evaluate the efficiency and effectiveness of our approach as well as alternative tuning options for different desired results. We have used a database containing information about movies taken from imdb[1]. In all cases, the input databases to the translator engine have been logical database subsets [15, 22] containing a relatively small number of relations (up to 8) and tuples (up to 1000).

We have used both quantitative and qualitative measures. The former include the *number of words* included in the narrative produced and the *execution time* for the translation to complete. The latter are subjective and include *readability*, capturing the extent to which a reader can easily understand a textual answer, and *concision*, capturing the extent to which a textual answer contains sufficient information with respect to the user expectations.

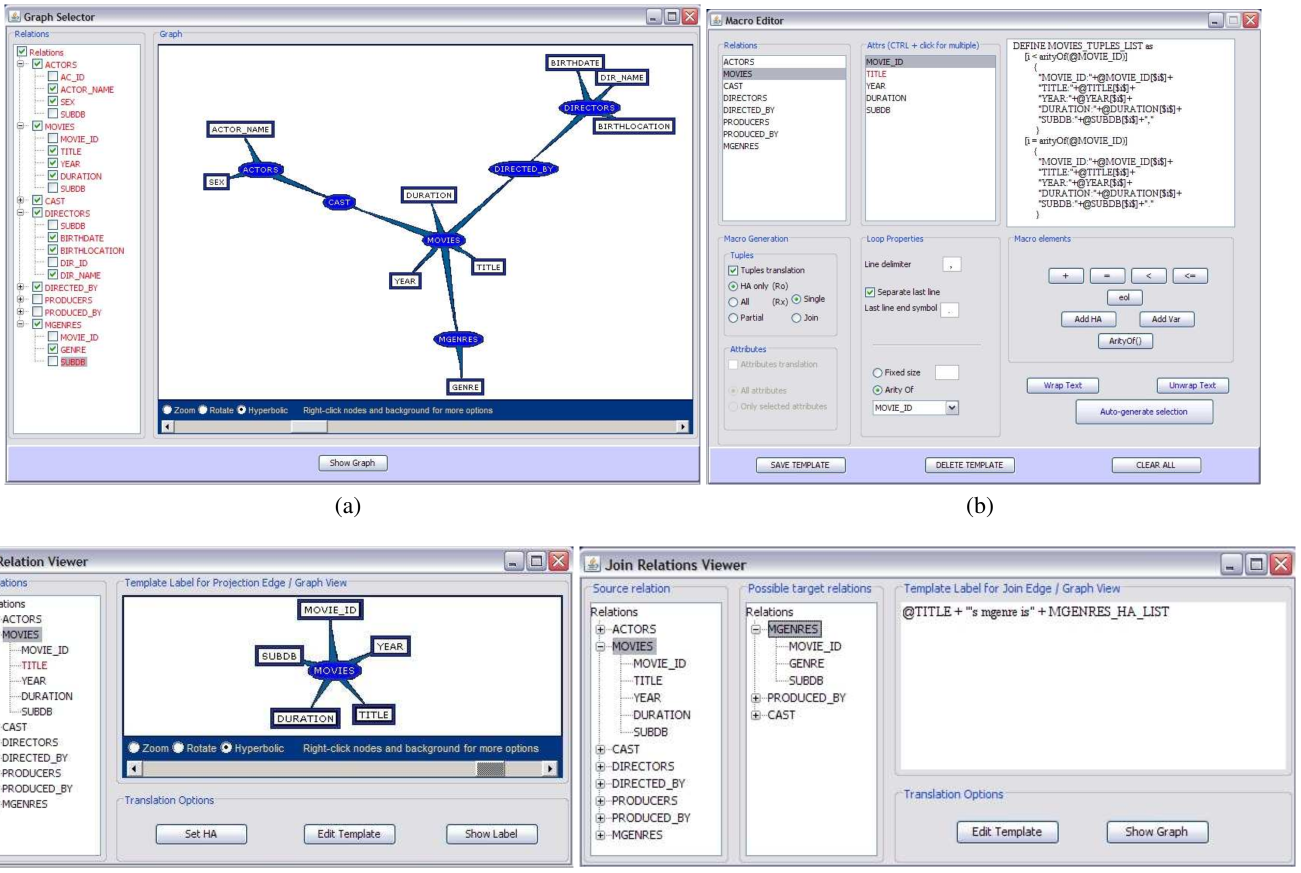
In the experiments, we have varied the following: (a) the method of database-graph traversal, i.e., either the *TRLS* algorithm (Figure 6) or a simpler version of it without any of the proposed mechanisms or the translation modules (referred to as *DFS*, as it resembles a straightforward translation following the depth-first search algorithm); (b) the absence or presence of the *resolve_common_expressions* mechanism (*RCE*); and (c) constraints either on the input database schema (*SC*) or on the cardinality of the input database (*CC*).

Regarding the *SC* constraints, we have considered two classes: soft (*sSC*) and hard (*hSC*) constraints. The former are not very strict on the size of phrases and on when those should be split (see Section 5.3), and allow more relations to be included in the result. In the experiments, as a soft constraint, we have considered that the distance of a relation from the initial one should be at most 3 joins away. On the other hand, the hard constraints restrict the size of phrases and favor the translation of relations that are close to the initial relation (2 or 3 joins away) in R^o mode, while the closest relations are translated in the mode imposed by *TRLS*. Regarding the *CC* constraints, we have examined two cases: the full-blown input database (no cardinality constraints, k=∞) and the k-limited tuple sets, i.e., only k tuples from each relation. (The translation favors tuples from a relation that can join with tuples from other relations and avoids the presence of nulls.) For k, we have used the values 1, 3, and 5. Note that, in the input logical database subsets, tuples are not ranked; otherwise, we would have top-k semantics.

In summary, we have experimented with the evaluation of quantitative and qualitative measures for the following cases (for k=∞, 5, 3, 1): (a) *DFS*; (b) *DFS* and *RCE*; (c) *TRLS* and *RCE*; (d) *TRLS*, *RCE*, *sSC*; and (e) *TRLS*, *RCE*, *hSC*. Figure 9(a) demonstrates the corresponding execution times. For translation with no *SC* constraints, the simple translation (*DFS*) and the proposed method (*TRLS*) have similar performance. The latter is slightly slower due to the additional operations performed. In the presence of such constraints, however, performance improves as smaller answers are produced. Use of *CC* constraints limits the result even further. Hence, as the cardinality constraints become stricter, k going from 5 down to 1, translation times decrease. In all cases, we observe reasonable execution times (less than 4.5 sec in the case of large logical subsets containing around 1000 tuples without constraints.) Execution times can be explained by Figure 9(b) that shows the number of words produced in each scenario and confirms that using (*SC* or *CC*) constraints reduces the answer size (and, thus, execution times).

Regarding qualitative experiments, a small number of users have provided feedback for a set of answers that corresponded to the

[1]www.imdb.com



```
DEFINE MOVIES_TUPLES_LIST as
    [i < arityOf(@MOVIE_ID)]
        {
         "MOVIE_ID:"+@MOVIE_ID[$i$]+
         "TITLE:"+@TITLE[$i$]+
         "YEAR:"+@YEAR[$i$]+
         "DURATION:"+@DURATION[$i$]+
         "SUBDB:"+@SUBDB[$i$]+","
        }
    [i = arityOf(@MOVIE_ID)]
        {
         "MOVIE_ID:"+@MOVIE_ID[$i$]+
         "TITLE:"+@TITLE[$i$]+
         "YEAR:"+@YEAR[$i$]+
         "DURATION:"+@DURATION[$i$]+
         "SUBDB:"+@SUBDB[$i$]+"."
        }
```

```
@TITLE + "'s mgenre is" + MGENRES_HA_LIST
```

(a) (b)

(c) (d)

Figure 8: Bird-eye view of our system

same set of queries. Figure 9(c) shows the evaluation in terms of the readability of the answers. Interestingly, although *DFS*, *DFS+RCE*, and *TRLS+RCE* have the same performance and outputs of similar size, readability of those outputs differs significantly. This is due to *RCE* and the translation modules incorporated in *TRLS*, which create textual output closer to the spoken language. Furthermore, we observe that readability increases with the application of (*SC* or *CC*) constraints, which coincides with the decrease of result size. Consequently, users found that smaller results are more understandable.

On the other hand, smaller answers may not always contain sufficient information regarding a user information need. Figure 9(d) presents the concision of the answers from the user perspective. There exist two factors that have driven the users' assessment: the quantity of information in the answer and the length of the answer. Interestingly, concision does not follow the same trends as readability. We observe that, for k = 3 or 5, answers have higher concision scores. Moreover, in all cases, the methods without *SC* constraints have almost identical scores. This is due to the absence of a discrimination criterion for the choice of tuples. If the tuples were considered in a ranked fashion (any ranking function can be used as input to the translator according to the application) then the results would be different for the cases where a *CC* constraint ($k \neq \infty$) is enforced. Figure 9(e) presents the average of readability and concision, weighing the two measures equally. One effect of taking readability into account is differentiation of the scores of methods without *SC* constraints.

Based on the above, *TRLS+RCE* and the enforcement of *SC* and *CC* constraints produce answers of acceptable quality but also of reduced size, which results in an efficient translation process. When the translation is to be used on the entire input database, then the use of *SC* constraints is highly recommended to control the output size and keep execution times low. If *CC* constraints exist, then the use of *TRLS* with or without *SC* constraints is suggested. We have also observed that some cases receive similar evaluations, e.g., for k=1 (Figure 9(d)). This is due to the absence of tuple ranking, which would provide room for differentiation.

7. CONCLUSIONS

In this paper, we have provided a framework and a method for the presentation of structured data as text. This is viewed as a proper structured management of individual tuples, according to certain rules and templates predefined by the designer or administrator of the database. We have described our prototype system and we have discussed several experiments that demonstrate how the narrative produced is affected as several features and parameters of our approach are varied, as well as how the response time of the system is affected by its configuration. We have also conducted a set of experiments measuring the impact that different answers, generated by different configurations, have on users.

As future work is concerned, we are interested in extending our approach towards explicitly taking into account tuple ranking in the text generated; e.g., either for producing texts, such as "... the most important movie of this director is ... Other movies include ..." or for generating shorter answers containing only the most interesting

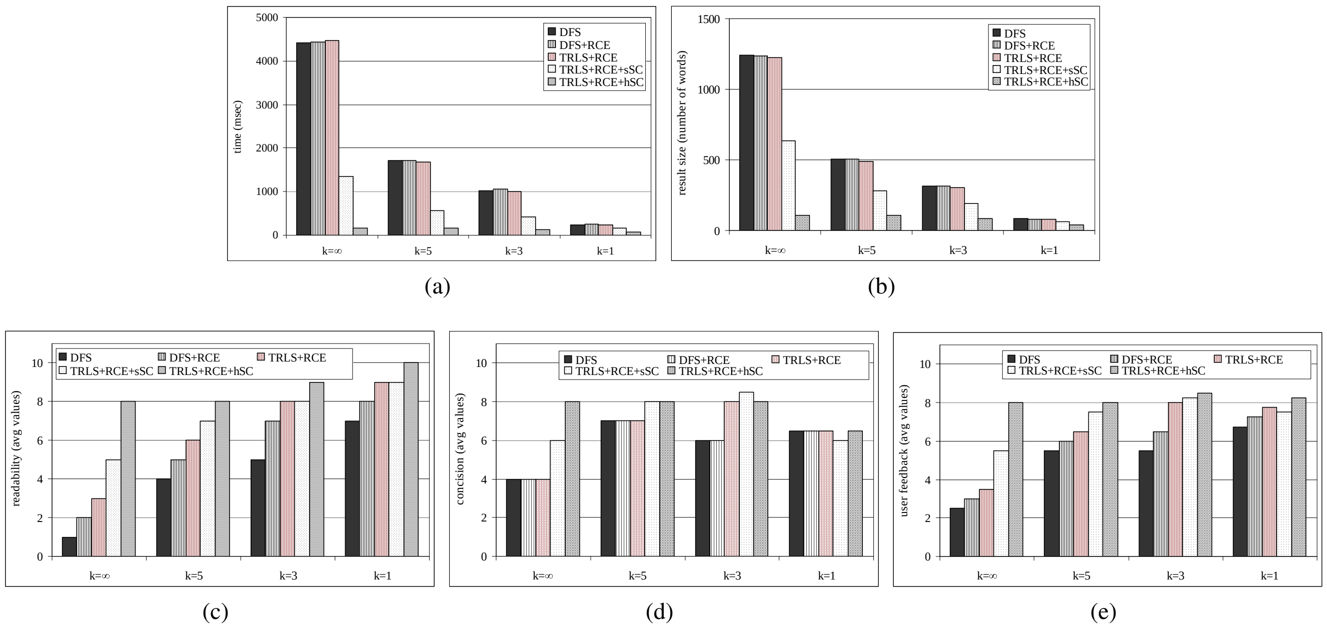


Figure 9: Evaluation results

information. Another challenge is to explore the feasibility of our approach with languages having a more complicated grammar than the English language.

Acknowledgments. We thank the anonymous referees for their careful reading of the paper and their valuable comments that significantly improved its presentation and quality.

8. REFERENCES

[1] www.diligentproject.org.

[2] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A system for keyword-based search over relational databases. In *ICDE*, pages 5–16, 2002.

[3] I. Androutsopoulos, G. D. Ritchie, and P. Thanisch. Natural language interfaces to databases - an introduction. *CoRR*, cmp-lg/9503016, 1995.

[4] AT&T Labs Inc. - Research. Text-to-speech (TTS), The synthesis of audible speech from text. url: http://www.research.att.com/ ttsweb/tts/index.php♯top, March 2007.

[5] Bell Labs. Text-to-speech synthesis. url: http://www.bell-labs.com/project/tts, March 2007.

[6] R. Cole, J. Mariani, H. Uszkoreit, G. Varile, A. Zaenen, V. Zue, and A. Zampolli. *Survey of the State of the Art in Human Language Technology*. Cambridge University Press and Giardini, 1997.

[7] P. Dixon. Basics of Oracle text retrieval. *IEEE Data Eng. Bull.*, 24(4):11–14, 2001.

[8] D. Florescu, D. Kossmann, and I. Manolescu. Integrating keyword search into XML query processing. *Computer Networks*, 33(1-6), 2000.

[9] J. Graupmann, R. Schenkel, and G. Weikum. The SphereSearch engine for unified ranked retrieval of heterogeneous xml and web documents. In *VLDB*, pages 529–540, 2005.

[10] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: Ranked keyword search over xml documents. In *SIGMOD Conference*, pages 16–27, 2003.

[11] J. R. Hamilton and T. K. Nayak. Microsoft SQL server full-text search. *IEEE Data Eng. Bull.*, 24(4):7–10, 2001.

[12] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient IR-style keyword search over relational databases. In *VLDB*, pages 850–861, 2003.

[13] V. Hristidis, Y. Papakonstantinou, and A. Balmin. Keyword proximity search on XML graphs. In *ICDE*, pages 367–378, 2003.

[14] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional expansion for keyword search on graph databases. In *VLDB*, pages 505–516, 2005.

[15] G. Koutrika, A. Simitsis, and Y. Ioannidis. Précis: The essence of a query answer. In *ICDE*, 2006.

[16] A. Maier and D. E. Simmen. DB2 optimization in support of full text search. *IEEE Data Eng. Bull.*, 24(4):3–6, 2001.

[17] E. Métais. Enhancing information systems management with natural language processing techniques. *Data Knowl. Eng.*, 41(2-3):247–272, 2002.

[18] E. Métais, J.-N. Meunier, and G. Levreau. Database schema design: A perspective from natural language techniques to validation and view integration. In *ER*, pages 190–205, 1993.

[19] M. Minock. A phrasal approach to natural language interfaces over databases. In *NLDB*, pages 333–336, 2005.

[20] T. Schultz and K. Kirchhoff. *Multilingual Speech Processing*, chapter 10. Speech-to-Speech Translation. Elsevier, Academic Press, April 2006.

[21] A. Simitsis and G. Koutrika. Comprehensible answers to précis queries. In *CAiSE*, pages 142–156, 2006.

[22] A. Simitsis, G. Koutrika, and Y. Ioannidis. Précis: From unstructured keywords as queries to structured databases as answers (to appear). *Int'l Journal on Very Large Data Bases*.

[23] E. Sneiders. Automated question answering using question templates that cover the conceptual model of the database. In *NLDB*, pages 235–239, 2002.

[24] V. C. Storey. Understanding and representing relationship semantics in database design. In *NLDB*, pages 79–90, 2000.

[25] V. C. Storey, R. C. Goldstein, and H. Ullrich. Naïve semantics to support automated database design. *IEEE Trans. Knowl. Data Eng.*, 14(1):1–12, 2002.

[26] A. Toral, E. Noguera, F. Llopis, and R. Muñoz. Improving question answering using named entity recognition. In *NLDB*, pages 181–191, 2005.

[27] Q. Wang, C. Nass, and J. Hu. Natural language query vs. keyword search: Effects of task complexity on search performance, participant perceptions, and preferences. In *INTERACT*, pages 106–116, 2005.