RELATIONAL QUERY FORMULATION
BY PSEUDONATURAL LANGUAGE TEXT MANIPULATION

Hirofumi Amano
Department of Computer Science and Communication Engineering
Kyushu University
Hakozaki, Higashi, Fukuoka 812, Japan

Yahiko Kambeysachi
Integrated Media Environment Experimental Laboratory
Kyoto University
Yoshida-Honmachi, Sakyo, Kyoto 606-01, Japan

ABSTRACT
This paper proposes an approach which allows users to edit natural language fragments given by the system and to obtain a pseudonatural language text describing the meaning of the intended query. Algorithms presented here translate manipulations on the pseudonatural language text being edited into changes on a query hypergraph as its internal representation, calculate possible quantifier scopes on the hypergraph, and translate the interpretation specified by the user into an SQL query.

1 INTRODUCTION
Relational database systems are now considered essential tools to store and process a large amount of data in office work, and widely used by many people including computer-novices. To get necessary information from those systems, however, one must specify to the system exactly what is needed. That request must be given in the way the system can understand. For this reason, formal query languages have been used most.

QBE is such a formal query language which is believed easy to learn. In an experiment, however, 27% of the QBE queries written by the subjects were semantically incorrect but syntactically correct. As long as a formal query language is used, the system has no way of knowing that a syntactically correct query is different from the user's intention.

To offer users an alternative to formal query languages, many attempts are being made to develop natural language interfaces. Natural language interfaces, however, also have their problems: natural language processing is not easy to learn. In an experiment, however, 27% of the QBE queries written by the subjects were semantically incorrect but syntactically correct. As long as a formal query language is used, the system has no way of knowing that a syntactically correct query is different from the user's intention.

A more modest approach is to implement a system which can accept only a reasonable variety of queries at the cost of rejecting other questions which are considered rare. Unfortunately, users must learn the language of the interface, and must know which sentences are allowed and which are not. This obscures the merit of using natural language.

Another problem in this approach is its expressive power. The coverage of queries of such a restricted language has often nothing to do with the coverage desirable to the application, or, is sometimes too small compared to that of formal query languages such as SQL. In most systems, for example, a simple (but also very useful in many applications) sentence like "Who supplies all the parts with color 'Red'" may be rejected just because the interface has not been tuned for this particular usage. This is due to the difficulty in handling quantifiers and aggregation nouns uniformly within such a simplified natural language translation approach.

In this paper, we present another modest approach, which allows users to edit natural language fragments given by the system and to formulate a pseudonatural language expression describing the meaning of the users' intention.

It extensively uses the structural similarity between hypergraph representations of relational queries and pseudonatural language expressions in the translation process. Therefore, while users edit a pseudonatural language query by assembling natural language fragments given by the system, the system can modify the internal representation of the query being formulated.

Quantifiers and aggregation nouns are also incorporated in this approach. Since they are formally treated in terms of hypergraphs and their components, the system need not be customized for ordinary expressions like all, only, no, or at least two, etc. All the system administrator must do for customization is to prepare a few more natural language fragments for each relation in the database. Pseudonatural language expressions employing quantifiers or aggregation nouns correspond to queries involving set comparisons or aggregation functions. These expressions are translated into various nested SQL queries by the algorithms presented in this paper.

2 BASIC CONCEPTS

2.1 Basic Relational Queries and Hypergraphs

A conjunctive SPJ query consists of a finite number of selection, projection, and join operations. Any given conjunctive SPJ query $Q_{SPJ}$ can be transformed into the following style:

$$Q_{SPJ} = \sigma_{P}(\pi_{x}((C_{1} \land C_{2} \land \ldots \land C_{m})))$$

where the symbols have the following meaning:
- $C_{1}$ is the conjunction of all the selection conditions;
- $C_{2}$ is the conjunction of all the join conditions;
- $P$ is the output attributes;
- $\sigma$, $\pi$, and $\times$ are selection, projection, and Cartesian product operators.

A hypergraph is a pair $(N, E)$ where $N$ is a finite set of nodes, and $E$ is a set of hyperedges which are arbitrary nonempty subsets of $N$. The union of all the hyperedges is equal to $N$.

A Berge cycle in hypergraph $H(N, E)$ is a sequence $(S_{1}, x_{1}, S_{2}, x_{2}, \ldots, S_{m}, x_{m}, S_{m+1})$ which satisfies the following conditions:

1. $m \geq 2$;
2. $x_{1}, x_{2}, \ldots, x_{m}$ are distinct nodes;
3. $S_{1}, \ldots, S_{m}$ are distinct hyperedges, and $S_{1} = S_{m+1}$;
4. $x_{i} \in S_{i}$ and $x_{i} \in S_{i+1}$ for $1 \leq i \leq m$.

A hypergraph is Berge-cyclic if it has a Berge cycle. Otherwise, it is Berge-acyclic. Figures 1(a) and 1(b) show examples of Berge-cyclic hypergraphs. Examples of Berge-cycles are indicated by dotted lines. A Berge-acyclic hypergraph is shown in Figure 1(c).
An object is a nonempty subset of a relation schema. Suppose that a set of objects is given for each relation, and that the union of each object set is equal to the relation schema. A query hypergraph for a given conjunctive SPJ query is a hypergraph such that:

1. its hyperedges are given by a set of objects which is minimal to cover the union of the selection, projection, and join attributes, where a pair of joined attributes with different names are renamed so that they share the same node;
2. each node $v_A$ representing an attribute $A$ which appears in a selection condition is labeled with the condition, such as $\alpha A \theta c$;
3. each node $v_B$ representing an attribute $B$ in $P$ is labeled as "output";

where $\theta$ is a scalar comparison operator and $c$ is a constant.

### 2.2 Query Language SQL

SQL has facilities for querying, data management and data definition. For the rest of this paper, however, we concentrate on its query facility. A typical SQL query construct, called a query block, is as follows.

```sql
SELECT << output_specification >>
FROM << relations >>
WHERE << condition >>
```

SQL allows a query block to be nested within a predicate in the WHERE clause of another query block. The inner block may contain yet another query block, and so on. In such a case, join predicates may involve attributes of the outer block or a higher block. A reference by such a join predicate is called an interblock reference.

We classify the nested predicates into the following six categories.

1. **Scalar comparison**: This form of nesting has an effect similar to that of a join operation, but is allowed only when the inner block returns only one value.

   $\ll \text{attribute} \gg \theta (\ll \text{query.block} \gg)$  \hspace{1cm} (2)

2. **Quantified scalar comparison**: This form is allowed when the inner block returns a relation with only one attribute.

   $\ll \text{attribute} \gg \theta \{\text{ALL | ANY | SOME}\} (\ll \text{query.block} \gg)$  \hspace{1cm} (3)

3. **Attribute-aggregation comparison**: This is similar to a selection operation, except that the constant is replaced by an aggregated value returned by the inner query block.

   $\ll \text{attribute} \gg \theta (\ll \text{aggregation.query.block} \gg)$  \hspace{1cm} (4)

   $\ll \text{aggregation.query.block} \gg$ is as follows.

   ```sql
   SELECT \{\{\text{MAX | MIN | COUNT | SUM | AVG}\} (\ll \text{attribute} \gg) | COUNT(*)\}
   FROM \ll \text{relations} \gg
   WHERE \ll \text{conditions} \gg
   ```

4. **Constant-aggregation comparison**: If there is no interblock reference to the outer block or a higher block, the value of this predicate has nothing to do with the result of the query, and the predicate is meaningless.

   $\ll \text{constant} \gg \theta (\ll \text{aggregation.query.block} \gg)$  \hspace{1cm} (5)

5. **Set membership checking**:

   $\ll \text{attribute} \gg \{\text{IN | NOT IN}\} (\ll \text{query.block} \gg)$  \hspace{1cm} (6)

   The inner block must return a relation with only one attribute.

6. **Existential checking**:

   $\{\text{EXISTS | NOT EXISTS}\} (\ll \text{query.block} \gg)$  \hspace{1cm} (7)

   If the predicate has no interblock reference to any outer block, it is meaningless.

### 3 PSEUDONATURAL LANGUAGE EXPRESSIONS AND CONJUNCTIVE SPJ QUERIES

This section summarizes the correspondence between conjunctive SPJ queries and pseudonatural language expressions.

Our pseudonatural language text generation strategy is based on a combination of several simple string manipulations, such as insertion of phrases and modifiers. Such natural language fragments are the units of these operations, and must be prepared by hand when the system is customized for a new application. The strategy is summarized as follows:

1. Determine the attributes which are relevant to a given query;
2. Assemble prepared natural fragments to form a natural-language-like sentence.

We must determine how to decompose attributes in a relation into several semantic units, and how to attach natural language fragments to those units. Furthermore, we need a set of simple transformation rules to generate natural-language-like sentences.

#### 3.1 Objects and Their Natural Language Fragments

To obtain suitable semantic units for string manipulation, we decompose a relation schema into several attribute sets. We call such an attribute set an object. An object is characterized by a simple natural language sentence in which each attribute in the object appears as a noun phrase. The sentence must not contain a noun phrase which refers to any attribute not of the object. If a natural sentence describing an object requires such an "external" attribute noun, the object should include that missing attribute.

Natural language fragments to be prepared are classified into two categories:
Canonical Sentences: Sentences which explicitly contain attribute names, and express the relationships among them.

Canonical Subclauses: Subclauses (relative clauses or prepositional phrases) which can be placed after noun phrases containing attribute names.

A canonical sentence expresses the relationship among the attributes in an object. However, combined descriptions may be necessary to express the meaning of queries, since they are likely to involve several relations and therefore several objects. The easiest way to construct such a combined sentence is to embed a prepared subclause for one object into a prepared sentence for another.

To simplify the transformation process, we introduce some modest restrictions on these natural language fragments. First, for all sentences associated with the objects of a given relation, we insist on a common subject. Second, those attribute names must appear explicitly in the sentences and clauses.

These natural language fragments are concerned only with the attributes of one object. This considerably decreases the customization work compared to the case where we have no localized units.

Example 3.1
Suppose that we have the following database schema:

SUPPLIER(S#, SNAME, ADDRESS);
PART(P#, PNAME, COLOR);
SUPPLY(S#, P#, QTY).

For relation SUPPLIER, we can now make the following sentences which have a noun phrase associated with S# as the common subject:

supplier with supplier-code { S#} : is called { SNAME} ;
: is located at {ADDRESS} ;

This gives us two objects {S#, SNAME} and {S#, ADDRESS}. The subclauses to be prepared are:

supplier called { SNAME} : which has supplier-code ( S#) ;
supplier located at {ADDRESS} : which has supplier-code {S#} ;

Note that the subclauses for S# can be made easily from the sentences above by inserting a relative noun.

3.2 Translating Conjunctive SPJ Queries into Pseudo-natural Language Expressions

Query hypergraphs, defined in Section 2, are intermediate representations for pseudonatural language text generation. We now need procedures to assemble natural language fragments to describe an SPJ query. Let S be the set of selection attributes which appear in the selection condition $C_S$ of Formula (1), and let J be the set of join attributes in $C_J$.

Algorithm 3.1 (Basic Text Generation)

Input: Attribute sets $S$, $P$, $J$;
Conditions $C_S$, $C_J$;
Objects of the database and their canonical expressions.

Output: A pseudonatural language expression describing the query.

Method: (Outline)

(1) Query Hypergraph Construction: Find an object set of minimum size which covers SUPPLIER. Let the object set be $O$. If the hypergraph for $O$ is Berge-cyclic, decompose it into a set of Berge-acyclic hypergraphs. In this case, the next three steps must be repeated for each Berge-acyclic hypergraph.

(2) Base Sentence Selection: Of the objects found in the previous step, choose one having the greatest number of attributes to be the generator of the base sentence. Let the object be $B$.

(3) Sentence Skeleton Construction: Find an object in $O$ whose hyperedge is connected to that of $B$, embed the subclause for the object into the sentence for $B$. If the object is associated with $B$ by a $\theta$-join (other than ‘=’), insert a phrase corresponding to the scalar comparison (such as “more than”, etc.). Repeat this until all the subclauses of objects in $O - \{B\}$ are embedded in the sentence.

(4) Sentence Modification: Insert a modifier phrase for each selection condition (such as “more than $c$”, etc.).

(5) Formatting: Emphasize output attributes and format the sentence(s).

In Step (1), the algorithm decomposes a Berge-cyclic hypergraph into a set of Berge-acyclic ones. In general, the relationships expressed in a sentence cannot be cyclic. Suppose that the hypergraph has a Berge cycle as in Figure 1(b). In such a case, the sentence would be lengthy, and at least one attribute would appear at both the head and the tail of the sequence of words. The decomposition instead makes two or more shorter sentences. Note that this decomposition does not rejects Berge-cyclic queries. It simply treats cyclic queries in a different manner.

Example 3.2
Suppose that we have the following query $Q_1$:

$$Q_1 = \pi_{\text{NAME}, P#, QTY}(\sigma_{QTY > 100}(\sigma_{\text{SUPPLIER.S#} = \text{SUPPLY.S#}}(\text{SUPPLIER} \times \text{SUPPLY})))$$

The relevant attributes are:

$$S \cup P \cup J = \{ \text{NAME, P#, QTY, S#} \}. \quad (9)$$

The covering object set $O$ is:

$$O = \{ \{ \text{NAME, S#} \}, \{ S#, P#, QTY \} \}. \quad (10)$$

$Q_1$ has the hypergraph shown in Figure 2. The base sentence chosen by the algorithm is:
supplier with supplier-code \(S\#\) supplies parts with part-code \(P\#\) in quantity \(QTY\).

The sentence skeleton is obtained by inserting the subclause for the object \(S\#, \ SNAME\):

\[
\text{supplier with supplier-code } \{S\#\} \text{ (who is called } \{SNAME\}\text{) supplies parts with part-code } \{P\#\} \text{ in quantity } \{QTY\}.
\]

Finally, \(Q_1\) is translated into the following pseudonatural language sentence:

\[
\text{List } \{SNAME\}, \{P\#\}, \{QTY\} \text{ such that:}
\]

\[
\text{supplier with supplier-code } \{S\#\} \text{ (who is called } \{SNAME\}\text{) supplies parts with part-code } \{P\#\} \text{ (called } \{PNAME\}\text{) in quantity } \{QTY\} \text{ (more than 100).}
\]

### 4 QUERY FORMULATION BY EDITING PSEUDONATURAL LANGUAGE EXPRESSIONS

Algorithms presented in the previous section translates conjunctive SPJ queries written in a formal query language into pseudonatural language expressions. These expressions themselves can be used for query verification by users.8,9,1)

When a particular query is translated into the corresponding pseudonatural language expression, however, the user may want to modify the query slightly. If a tailored editor can be constructed which guides a user to formulate a legal pseudonatural language expression only and the system can modify the original query following the operations the user performed, it will be useful for: (a) incremental refinement of queries; (b) query formulation on pseudonatural language expressions from scratch.

All that the editor must do is to accept simple manipulations on the pseudonatural language expression displayed on the screen, and to translate them into changes on the internal form without showing them to the user. If quantifiers and aggregation nouns can be used, the user can formulate various patterns of complex queries.

Quantification, however, leads to an important problem: multiple interpretations in quantifier scope. Automatic disambiguation would require situational information or the knowledge about the world, other than the knowledge about the language in use. This is not in the scope of this paper. The basic strategy presented in this section is instead to present those interpretations to the user and to let the user choose the one in his/her mind. After the user chooses the interpretation which fits his/her intention, all that the system must do is to translate it into a query executable on the target database.

#### 4.1 Editing Conjunctive SPJ Queries through Pseudonatural Language Expressions

Suppose that a user is given a pseudonatural language expression displayed on the screen. After moving a cursor to an arbitrary position on the displayed expression, the user may edit it slightly, guided by the system.

Since the structural correspondence between conjunctive SPJ queries and query hypergraphs is obvious, the system can modify the internal query hypergraph automatically.

Let us exhibit the process by an example.

**Example 4.1 (Editing Conjunctive SPJ Queries)**

Suppose that the user has the pseudonatural language expression in Example 3.2. Now, suppose that the user performs the following edit operations:

1. locating the cursor at \("P\#"\) and inserting a subclause \("(called \{PNAME\})"\) from the choices the system displays;
2. locating the cursor at \("PNAME\)" and replacing it with \("Bolts\); \n
These operations gives the following pseudonatural language expression:

\[
\text{List } \{SNAME\}, \{P\#\}, \{QTY\} \text{ such that:}
\]

\[
\text{supplier with supplier-code } \{S\#\} \text{ (who is called } \{SNAME\}\text{) supplies parts with part-code } \{P\#\} \text{ (called } \{PNAME\}\text{) in quantity } \{QTY\} \text{ (more than 100).}
\]

These operations are converted to the following modifications on the hypergraph representation in Figure 2:

1. adding a hyperedge \(\{P\# , PNAME\}\), and put a label \("SUPPLY.P#=PART.P#\) to node \("P\#\);
2. putting the label \("PNAME='Bolts'\) to node \("PNAME\).\n
The modified hypergraph is shown in Figure 3.

With the help of the correspondence between query hypergraphs and conjunctive SPJ queries, this hypergraph can be translated into a new query \(Q_2\) in the form of Formula (1).

\[
Q_2 = \pi_{\{SNAME, P\#, QTY\}}(\sigma_{\{QTY>100 \land PNAME='Bolts'\}}(\delta_{SUPPLIER.S#=SUPPLIER.S#}(SUPPLY.P#=PART.P#)SUPPLIER \times SUPPLY \times PART))).
\]

**Figure 3: The Query Hypergraph of Query \(Q_2\)**

#### 4.2 The Scoping Problem of Quantifiers in Pseudonatural Language Expressions

In this paper, a quantifier is defined as follows:

A word or phrase which is used with an attribute noun phrase, and which shows either: (a) the cardinality of the set of values for the attribute; or (b) the relative size of the set of values for the attribute with respect to the specified domain.
This is to distinguish quantifiers such as “all”, “only”, etc. from other descriptions such as “many”, “few”, etc.; or “three kilograms of”, etc.\(^5\)

Suppose that a user inserts a natural quantifier such as “all”, “only”, “no”, or “at least \(<\neq n \geq\)”. The choices of natural quantifiers should be given by the system. If the system is equipped with plural forms for attribute nouns, it is not difficult to replace a singular form with a plural. If the system can translate the quantified pseudonatural language expression into a formal query, the user can formulate and execute the query without seeing it.

In QBE, queries in this class may involve quantifier “ALL” or aggregation function. An experiment showed that each of these features caused 20\% of the errors made by the subjects.\(^\text{13}\)McLeod also pointed out that the interpretation of a QBE query is not obvious in cases where it contains multiple set-oriented constructs.\(^\text{12}\) Allowing novice users to edit such queries through pseudonatural language expressions thus can be very useful.

Unfortunately, quantified expressions in natural language have ambiguities in quantifier scope. Suppose that we have the following sentence:

*Every supplier supplies a red part.*

It may mean that every supplier has its own red part to supply. It is, however, logically possible that there exists a red part supplied by all suppliers without exception.

One solution\(^\text{15}\) is given by using the following format:

\[
\text{For } \text{<<natural-quantifier>> } \text{<<variable>>} \text{ which satisfy } \text{<<domain-specification>>}
\]

\[
\text{<<restriction-on-the-domain>> .} \quad (12)
\]\n
We adopt this format for displaying the possible interpretations for a given quantified pseudonatural language expressions in an unambiguous way.

The general format of an unambiguously-quantified pseudonatural language expression is:

\[
\text{For } \text{<<quantifier>> } \text{<<quantified-noun>>}
\]

\[
\text{<<subclause-for-domain-specification>>,} \quad \text{<<restriction-on-the-domain>> .} \quad (13)
\]

In the rest of this paper, this format is referred to as the quantification normal form.

### 4.3 Disambiguation of Quantifier Scopings

The basic strategy for disambiguation is as follows:

1. Find all permissible scopings on the hypergraph;
2. Translate them into pseudonatural language expressions in the quantification normal form, and show them to the user;
3. Let the user choose the one that he/she has in mind, and translate it into an SQL query.

**Assumption 4.1 (Quantifier Scopes)**

A *scoping* of a quantified pseudonatural language expression is the classification of the nodes in the query hypergraph such that:

1. The domain specification part of the hypergraph is:
   - (i) a single node for the quantified attribute, or
   - (ii) a connected subhypergraph containing that node;
2. The restriction part of the hypergraph is a connected, non-empty subhypergraph containing the node for the quantified attribute;
3. The domain specification part and the restriction part share only the node for the quantified attribute.

Assumption (1) describes that the domain is either: (i) the set of all possible values for the attribute; or (ii) the values specified by the subhypergraph.

The next assumption just assures that the query has a meaning. The last assumption assures that a scoping can be described by a plain pseudonatural language expression in the quantification normal form. There is no correlation between the restriction part and the domain specification. The noun phrase for each attribute appears in only one position in the expression. This assumption may eliminate some of the logically possible interpretations. This is, however, necessary to keep its pseudonatural language translation as plain as possible. It is, of course, possible to let the user add a subclause corresponding to correlations between the two parts.

**Algorithm 4.1 (Quantifier Scoping)**

**Input:** A query hypergraph containing the label indicating node \(q\) for the quantified attribute;  

**Output:** The quantifier scopings on a query hypergraph;  

**Method:** (Outline)

1. If \(n\) distinct connected components share only \(q\), the scopings are \(2^n - 1\) combinations indicating which component belongs to which part, except the one with an empty restriction part.
2. Otherwise, \(q\) is the domain specification part, and the restriction is represented by the whole hypergraph.

**Example 4.2**

Suppose that the user formulates the following expression:

*Supplier with supplier code \([S\#]\) (called \(SNAME\)) supplies all types of parts (called ‘Bolts’)*;

where “types of parts” should be prepared beforehand as a plural. The query hypergraph is shown in Figure 4. For this hypergraph, Algorithm 4.1 generates the three scopings in Figure 5.

**Algorithm 4.2 (Text Generation with a Quantifier)**

**Input:** A query hypergraph and its scoping  

**Output:** A pseudonatural language expression in Format (13);  

**Method:** (Outline)

1. Put the quantifier and the quantified noun phrase in the template.
2. If the domain specification part is a single node. Omit “<<subclause-for-domain-specification>>”.
If the domain specification part is a nonempty hypergraph, translate this part using Steps (3) and (4) of Algorithm 3.1. Replace "<subclause_for_domain_specification>" with it.

(4) If the domain specification contains an output attribute, enclose it with "[ ]" instead of { }.

(5) Translate the restriction part by Algorithm 3.1, except that the quantifier must be replaced with "those".

Example 4.3
For the scopings given in Example 4.2, Algorithm 4.2 generates the following pseudonatural language expression for the scoping in Figure 5(a):

For all types of parts,
supplier with supplier-code [S#] (called [SNAME]) supplies those parts (called 'Bolts').

The scoping in Figure 5(b) has a nonempty hypergraph for the domain specification part. It is translated into the following expression:

For all types of parts (those which are called 'Bolts'),
supplier with supplier-code [S#] (called [SNAME]) supplies those parts.

The scoping in Figure 5(c) is translated into:

For all types of parts (those which are supplied by supplier with supplier-code [S#]),
those parts are called 'Bolts'.

4.4 Translation of the Chosen Interpretation into SQL
After the user chooses the interpretation which fits his/her intention, all that the system must do is to translate it into a query executable on the target database.

Natural quantifiers introduced in Subsection 4.2 can be expressed by set comparisons or intersection counts involving the set defined by the domain specification part and the set defined by the restriction part.

Example 4.4
When "all" is used in Format (13), this expresses that the set determined by the domain specification is contained in the set determined by the restriction. Let the former set be Dom, and the latter be Res. Then, this operation is expressed as:

$$\text{Dom} \subseteq \text{Res}. \quad (14)$$

Some other quantifiers and their set-oriented interpretations are also shown in Table 1.
Figure 6: The Template for Quantifier "all"

Figure 7: The Template for Quantifier "no"

Figure 8: The Template for Quantifier "at least <n>"
Quantifiers and aggregation nouns are also incorporated in this approach. These expressions correspond to complex SQL queries employing nested predicates, which are difficult to learn and use for non-expert SQL users. Since they are formally treated in terms of hypergraphs and their components, the system need not be customized for ordinary quantifications such as "all" or "at least n". This means, if the system is properly equipped with a small set of application specific natural language fragments, users can handle a certain set of complex SQL query patterns without seeing the actual SQL source text.

Quantifiers and aggregation nouns are also incorporated in this approach. These expressions correspond to complex SQL queries employing nested predicates, which are difficult to learn and use for non-expert SQL users. Since they are formally treated in terms of hypergraphs and their components, the system need not be customized for ordinary quantifications such as "all" or "at least n". This means, if the system is properly equipped with a small set of application specific natural language fragments, users can handle a certain set of complex SQL query patterns without seeing the actual SQL source text.

Quantifiers and aggregation nouns are also incorporated in this approach. These expressions correspond to complex SQL queries employing nested predicates, which are difficult to learn and use for non-expert SQL users. Since they are formally treated in terms of hypergraphs and their components, the system need not be customized for ordinary quantifications such as "all" or "at least n". This means, if the system is properly equipped with a small set of application specific natural language fragments, users can handle a certain set of complex SQL query patterns without seeing the actual SQL source text.

Quantifiers and aggregation nouns are also incorporated in this approach. These expressions correspond to complex SQL queries employing nested predicates, which are difficult to learn and use for non-expert SQL users. Since they are formally treated in terms of hypergraphs and their components, the system need not be customized for ordinary quantifications such as "all" or "at least n". This means, if the system is properly equipped with a small set of application specific natural language fragments, users can handle a certain set of complex SQL query patterns without seeing the actual SQL source text.

Universal relations are studied as a user interface in which users need not specify relation names. All that users must do is to state the selection conditions and the output attributes. This facility is achieved at the cost of restricting the possible join paths to those specified in or deduced from the object structure. Intuitively, such queries correspond to single SQL query blocks without nesting.

Instead, our approach makes the user specify the join path step by step. Though it may be slightly inconvenient for users, this scheme works even in the case where the user takes irregular join patterns which are not expected by the database designers. Moreover, quantified expressions can be processed in our approach, and translated into complex nested SQL queries. Such queries cannot be handled in the universal relation interface.

This approach can be useful in a cooperative environment. When non-expert users start working on queries written by other people, the pseudonatural language text generation algorithms\(^1\) can give easy-to-read paraphrases for those queries. Since the algorithms in this paper allow the new users to modify those queries on the pseudonatural language expressions, the effort required to customize stored queries can be drastically reduced.

**REFERENCES**


