Translation of SQL Queries Containing Nested Predicates into Pseudonatural Language

Hirofumi Amano

Dept. Computer Science and Communication Engineering
Kyushu University
Hakozaki, Higashi, Fukuoka 812, Japan

Yahiko Kambayashi

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

Abstract

An approach to support query verification by translating SQL queries into easy-to-read pseudonatural language expressions is proposed. The method discussed here converts various types of SQL queries, including simple and composite nested ones. Since it employs no complex natural language processing technique, it is feasible even on small computers.

1 Introduction

Recent advances in computers and databases have enabled computer novices to use commercially-supplied relational database systems on their own computers. Relational databases are no longer special tools for expert users only.

One of the criteria by which the novice users may choose a database system is the effort required each time they retrieve information from the database. To get necessary information, one must specify to the system exactly what is needed. For this purpose, formal query languages have been used most. Such an artificial language is easy to understand for the system, but not necessarily for novice users.

QBE[13] is a formal query language for relational databases which is believed to be easy to learn. In an experiment[11], 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 novice users an alternative to formal query languages, many attempts are being made to develop natural language interfaces which accept queries written in everyday language. Natural language interfaces, however, also have their problems. First of all, natural language processing is not an easy task. Our everyday language has too many rules and exceptions compared with formal languages. If we are to develop an interface which can accept every naturally-expressed input, and can respond in a natural way, we would have to equip the system with a huge store of human knowledge.

A more modest approach is to implement a system which can accept only a reasonable variety of queries at the cost of rejecting other queries which are considered rare. It would need only a limited amount of lexical, syntactic, situational or application-specific information. Unfortunately, *users must learn the language* of the interface, and must know which sentences are allowed and which are not. Such an interface could hardly be considered to be truly natural[4].

We can adopt another approach to cope with the problems of formal query languages. If users can verify the semantics of their queries, those possibly-incorrect queries will not be executed, and thus users will not misunderstand the information returned by the system. Natural language can be used for a feedback to the users[6, 7, 3, 9, 10]. They can check the output given by the system easily, and can execute the query only after they are sure that it fits their intention. This seems to be a reasonable compromise given the current technology.

The REMIT system[9] translates relational algebra queries into natural language. It uses a semantic network for representing application specific information. This network representation, however, may not be convenient in cases where the database structure is likely to change, since a small change in the network may cause a rippling effect throughout the network. This system also uses complex natural language processing techniques.

The ELFS system[10] has an preprocessor which transforms SQL[2] queries into an SQL-like intermediate language. Then, its translator generates a natural language text describing the meaning of the query. This translator employs simple binary relationships to store application-specific information. Binary relationships, however, are not powerful enough for handling n-ary ones in a natural way.

A third approach [7] employs hypergraphs and natural language fragments for representing application specific information. This hypergraph scheme can express n-ary relationships easily. For each element in those hypergraphs, a natural language fragment is assigned. Since these units are localized within each relation which is a unit in a whole database, this significantly reduces the effort required to customize the system for particular applications. The method first translates a given relational algebra query into a query hypergraph. This hypergraph is then

DATABASE SYSTEMS FOR ADVANCED APPLICATIONS '91
Ed. A. Makinouchi

used to determine a "skeleton" of the output sentence. Prepared natural language fragments are assembled into a *pseudonatural language expression* by simple string manipulations. The output is a natural-language-like expression describing the meaning of the result to be retrieved by the query.

In this paper, we extends this third approach to handle more complex queries written in SQL. An interesting, but also troublesome, feature of SQL is its nesting facility. In a complex query involving set comparisons or aggregations, a unit construct of SQL is embedded into another. Unfortunately, such queries are difficult to understand. This paper discusses a translation method for such nested SQL queries, which will be incorporated into a prototype system of an example-based natural-language-assisted interface[6] now under development[3].

2 Basic Concepts

2.1 Relational Databases and SPJ Queries

A *relation* which can be visualized as a flat table. The name of a column of such a table is an *attribute* and is used to identify a certain column in the table. Each row of the table corresponds to an element of the relation, called a *tuple*. The set of attributes of a relation R is called the *relation schema*, denoted $\boldsymbol{R}$. Values in a tuple t for an attribute set X is denoted $t[X]$. As long as there is no ambiguity, we use a concatenation of attribute names to refer to a set of attributes (such as AB for $\{A, B\}$).

A *projection* of a relation R onto an attribute set X, denoted $\pi_X(R)$, is defined as follows:

$$\pi_X(R) \stackrel{\text{def}}{=} \{r[X] \mid r \in R\}, \tag{1}$$

where $X \subseteq \boldsymbol{R}$.

Let θ be a scalar comparison operator, and c be a constant. A θ-selection of a relation R with the condition $A\theta c$, denoted $\sigma_{A\theta c}(R)$, is defined as follows:

$$\sigma_{A\theta c}(R) \stackrel{\text{def}}{=} \{r \mid (r[A]\theta c) \wedge (r \in R)\}, \tag{2}$$

where $A \in \boldsymbol{R}$. The scalar comparison between A and c is called a *selection condition*. A selection condition can be a conjunction of similar scalar comparisons.

The *θ-join* of two relations R and S with a condition $A\theta B$, denoted $R \underset{A\theta B}{\bowtie} S$, is defined as follows:

$$R \underset{A\theta B}{\bowtie} S \stackrel{\text{def}}{=} \{t \mid (t \in (R \otimes S)) \wedge (t[A]\theta t[B])\}, \tag{3}$$

where $A \in \boldsymbol{R}$, $B \in \boldsymbol{S}$, and "$\otimes$" denotes Cartesian product. The scalar comparison between A and B is called a *join condition*. A join condition can be a conjunction of similar scalar comparisons. A and B are called *join attributes*.

An *SPJ query* on a given database is a sequence of a finite number of selection, projection, and join operations. Any given SPJ query Q_{SPJ} can be transformed into the following style:

$$Q_{SPJ} = \pi_P(\sigma_{C_S}(\sigma_{C_J}(R_1 \otimes R_2 \otimes \cdots \otimes R_n))), \tag{4}$$

where C_S is the conjunction of all the selection conditions, C_J is the conjunction of all the join conditions, and P is the output attributes of the query Q_{SPJ}.

2.2 Hypergraphs

A *hypergraph*[1] 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 *path* from node x to y is a sequence of $k(\geq 1)$ hyperedges $E_1, \ldots, E_k$ such that: (a) $x \in E_1$; (b) $y \in E_k$; (c) $Ei \cap E_{i+1} \neq \emptyset$ for $1 \leq i \leq k$.

The above sequence $E_1, \ldots E_k$ may be called an *edge path* from E_1 to E_k. Two nodes are *connected* if there is a path from one to the other. Two hyperedges are connected if there is an edge path from one to the other.

A *Berge cycle* in hypergraph $H(N, E)$ is a sequence $(S_1, x_1, S_2, x_2, \cdots, S_m, x_m, S_{m+1})$ which satisfies the following conditions:

(a) $m \geq 2$;

(b) $x_1, x_2, \ldots, x_m$ are distinct nodes;

(c) $S_i, \ldots, S_m$ are distinct hyperedges, and $S_1 = S_{m+1}$;

(d) $x_i \in S_i$ 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*.

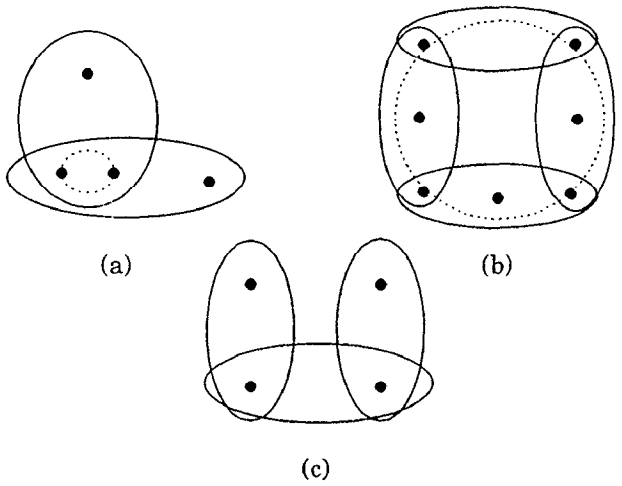


Figure 1: Examples of Hypergraphs

Figure 1(a) and (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).

2.3 Query Language SQL

For the rest of this paper, we concentrate on a typical SQL[2] query construct, called a *query block*. A query block is as follows.

SELECT $\ll$ *output_specification* $\gg$
FROM $\ll$ *relations* $\gg$
WHERE $\ll$ *condition* $\gg$

The WHERE clause is optional, while the SELECT and FROM clauses are mandatory. We assume that the WHERE clause may contain a conjunction of: (a) a conjunction of selection and join conditions (predicates) corresponding to those which appeared in Formula (4); or (b) a logical formula[1] of nested predicates (to be summarized later).

Intuitively, the above construct is interpreted as:

> *Calculate the Cartesian product of the relations specified in the FROM clause, then get only the tuples which satisfy the condition in the WHERE clause. Finally, print the values, following the output specification in the SELECT clause.*

Note that without aggregation in the output specification, a single query block can express an SPJ query.

SQL allows a query block to be nested in a predicate of the WHERE clause for another query block. The inner block may contain yet another query block, and so on. In such cases, join predicates may involve attributes of relations in the FROM clause of the outer block or a higher block[2]. A reference by such a join predicate is called an *interblock reference.*

The single-level nested predicates considered in this paper are classified into the following six categories[3] according to the linkage expressed by the predicates.

(a) **Category-S (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:

$$\ll attribute \gg \theta\ (\ll query\_block \gg) \qquad (5)$$

(b) **Category-Q (Quantified scalar comparison):**
This form is allowed when the inner block returns a relation with only one attribute:

$$\ll attribute \gg \theta\ \{\text{ALL} \mid \text{ANY} \mid \text{SOME}\}\ (\ll query\_block \gg). \qquad (6)$$

(c) **Category-A (Attribute-aggregation comparison):**
This is similar to a selection condition, except that the constant is replaced by an aggregated value returned by the inner query block:

$$\ll attribute \gg \theta\ (\ll aggregation\_query\_block \gg), \qquad (7)$$

where $\ll aggregation\_query\_block \gg$ is as follows.

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

(d) **Category-C (Constant-aggregation comparison):**
The general form of a Category-C nested predicate is:

$$\ll constant \gg \theta\ (\ll aggregation\_query\_block \gg). \qquad (8)$$

This compares the constant with the aggregated value returned by the inner block. 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.

(e) **Category-M (Set membership checking):** This checks whether a certain value of a tuple is contained in the set returned by the inner block. The inner block must return a relation with only one attribute. A general formulation is:

$$\ll attribute \gg\ \{\text{IN} \mid \text{NOT IN}\}\ (\ll query\_block \gg). \qquad (9)$$

(f) **Category-E (Existential checking):** A Category-E nested predicate checks whether or not the set returned by the inner block is empty. If the predicate has no interblock reference to another block above it, it is meaningless for the same reason as for Category-C. A general form is:

$$\{\text{EXISTS} \mid \text{NOT EXISTS}\}(\ll query\_block \gg). \qquad (10)$$

To clarify the semantics of nestings, we decompose each category into several types.

The first subcategorization is according to interblock references. If the inner block refers to its parent block, we distinguish that nesting type from the one without such a reference by underlining the category name. For example, a Category-S nesting with a one-level interblock reference is Type-$\underline{S}$.

The other subcategorization is by negations allowed in Category-M and Category-E. A negative nesting in those categories is denoted with a superscript "$-$", such as Type-M^- or Type-E^-.

A *nesting graph* for an SQL query is a directed graph (N, E) such that:

(a) Each block in the query is expressed as a node.

(b) If a block involves only one inner block, the nodes for the two blocks are connected by an edge labeled with the nesting type. For a logical formula of nested predicates, a conjunction of terms is expressed as a tree which has an "AND" node for its root and subtrees for the terms. Similarly, a disjunction is a tree whose root is an "OR" node. The logical formula thus comprises a tree structure.

[1] We assume that this formula has already been transformed so that negations appear only at the literal level, that is, at the individual predicates.

[2] We assume that a join predicate can involve at most one "external" attribute.

[3] Though Kim gave a different classification of nested predicates (in an old syntax of SQL)[8], that work was done from a viewpoint of query optimization, and would not suffice for our purpose.

(c) An interblock reference is expressed as an edge between the node for the block containing the join predicate and the node for the block referred to, labeled J (which stands for "Join").

(d) Each edge has the direction from the lower level to the higher level.

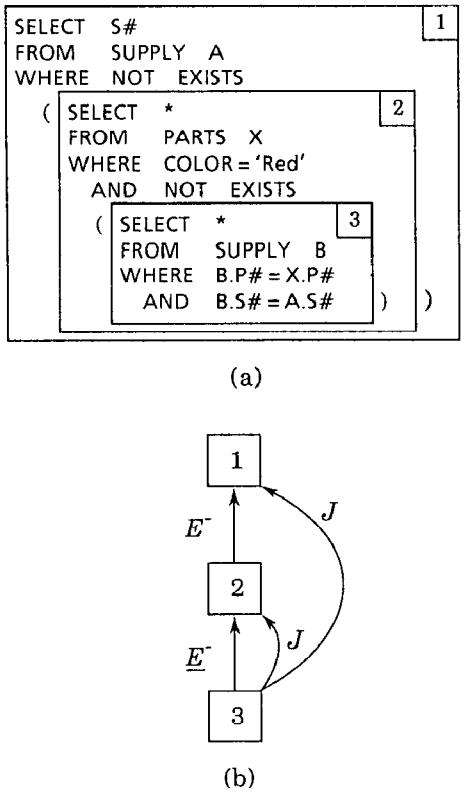


Figure 2: An Example of a Nested SQL Query and Its Nesting Graph

Example 1 : The nested SQL query in Figure 2(a) has the nesting graph shown in Figure 2(b). The boxes in Figure 2(a) indicate boundaries of query blocks.

3 Pseudonatural Language and SPJ Queries

This section summarizes an approach[7] to convert a class of basic relational queries, SPJ queries, into pseudonatural language.

Our text generation strategy is based on a combination of several simple string manipulations, such as insertion of phrases and modifiers. Natural language fragments must be prepared when the system is customized for a new application. The basic strategy is summarized as follows:

(a) Determine the attributes which are relevant to a given query;

(b) Assemble prepared natural fragments to form a natural-language-like sentence.

The first subsection discusses the natural language expressions to be used in pseudonatural language text generation. Then, we consider how to generate a pseudonatural language sentence describing a given SPJ query in the next subsection.

We assume that a given SPJ query has already been transformed into the form of Formula (4). This causes no loss of generality[4]. Let S be the set of selection attributes which appear in the selection condition C_S of Formula (4), and let J be the set of join attributes in C_J.

3.1 Objects and Their Natural Language Expressions

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[5].

For each object, we construct natural language fragments to be used as building blocks for text generation. These expressions 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 express the relationship between the attributes in an object. However, combined descriptions may be necessary to express the meaning of SPJ 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. This avoids repeated appearances of the same attribute in the case where several objects of one relation are involved in the query. 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 2 : Suppose that we have the following database schema:

[4]Note that this query transformation is just for pseudonatural language translation.

[5]If a natural sentence describing an attribute set requires such an "external" attribute noun, the set should include that missing attribute to form an object.

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 easily be made from the sentences above by inserting a relative noun.

3.2 Translating SPJ Queries into Pseudonatural Language Expressions

The natural language fragments discussed in the previous subsection are localized within a relation. We now need procedures to assemble those fragments to describe an SPJ query involving more than one relation.

Outline of the Translation Algorithm for SPJ Queries (Algorithm 1)

Input: Attribute sets S, P, J;
Conditions C_S, C_J;
Objects of the database and their canonical expressions.
Output: A natural language expression describing the query.
Method:

(a) **Hypergraph Construction**: Find an object set of minimum size which covers $S \cup P \cup J$. Let the object set be $\boldsymbol{O}$. If the hypergraph for $\boldsymbol{O}$ 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.

(b) **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.

(c) **Sentence Skeleton Construction**: Find an object in $\boldsymbol{O} - \{B\}$ 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 θ-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 $\boldsymbol{O} - \{B\}$ are embedded in the sentence.

(d) **Sentence Modification**: Insert a modifier phrase for each selection condition (such as *"more than c", etc.*).

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

In step (a), 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 the head noun of the subject would appear at both the head and the tail of the sequence of words. The decomposition instead makes two or more shorter sentences having no repetition of the subject.

Step (b) chooses the "largest" object as the generator of the sentence and thus avoids to use it as a subclause. This is because such a "large" object is likely to have a long natural language string describing it and a shorter subclause is preferable to a longer one.

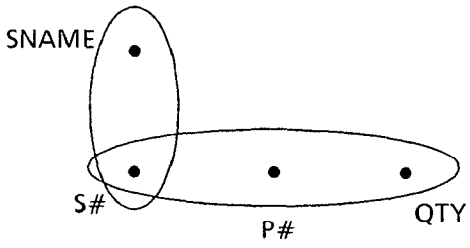


Figure 3: The Hypergraph of Query Q_1

Example 3 : Suppose that we have the following query Q_1:

$$Q_1 = \pi_{\{SNAME, P\#, QTY\}}(\sigma_{QTY>100}(\sigma_{SUPPLIER.S\# = SUPPLY.S\#}(SUPPLIER \otimes SUPPLY))) \quad (11)$$

The relevant attributes are:

$$S \cup P \cup J = \{SNAME, P\#, QTY, S\#\}. \quad (12)$$

The covering object set $\boldsymbol{O}$ is:

$$\boldsymbol{O} = \{\{SNAME, S\#\}, \{S\#, P\#, QTY\}\}. \quad (13)$$

Q_1 has the hypergraph shown in Figure 3. 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}:

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

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

List [SNAME], [P#], [QTY] such that:
supplier with supplier-code {S#} (who is called [SNAME]) supplies parts with part-code [P#] in quantity [QTY] (more than 100).

4 Nested SQL Queries and Their Pseudonatural Language Expressions

This section discusses our translation method for SQL queries containing such nested predicates. In the first subsection, the algorithm of Section 3 is slightly modified to cope with single nested predicates in Categories-S and -A. The second subsection analyzes other single nested predicates to be expressed using quantified expressions in natural language, and presents an approach to incorporate such quantified expressions into pseudonatural language. The last subsection introduces heuristics to translate SQL queries containing composite nested predicates, using the above results.

4.1 Single Nested Predicates in Categories-S and -A

As summarized in Section 2, single nested predicates in Categories-S and -A have semantics similar to SPJ queries.

(a) Modifications to Algorithm 1 for Category-S Nestings

Two-block queries in Category-S are special cases of SPJ queries. They can be translated into pseudonatural language by Algorithm 1, after obtaining a hypergraph for each query block and join the two hypergraphs. In practice, a warning concerning the restraint on the result of the inner block should be issued to the user.

(b) Modifications to Algorithm 1 for Category-A Nestings

A query with a single nested predicate in Category-A has two query blocks whose hypergraphs cannot be joined directly. The inner block, however, returns an aggregated value which is to be compared with an attribute in the outer block. Since this is similar to a selection operation, we first translate the outer block by Algorithm 1. At this point, selection constants should be replaced by a noun phrase such as "*the maximum of* $\ll$*noun_for_aggregated_attribute*$\gg$ *for*." The aggregate nouns, such as *minimum, average* etc., must be built into the system.

The inner block should be a sequence of noun phrases for the attributes determining the set of values on the aggregated attribute.

(c) Modifications to Algorithm 1 for Nested Predicates in Category-C except COUNT Function

If aggregate nouns are supported by the system, the algorithm can be applied also to all types of Category-C nestings except those using the **COUNT** function (Category-C nesting using **COUNT** is discussed in the next subsection). We simply translate the outer block, and add additional descriptions of the comparison between the constant and the aggregated value. The aggregated value can be paraphrased in the same way as with Category-A.

Type	Parapharasing Strategy
$S, \underline{S}$	Use the same algorithm as SPJ queries. Add the warning about the number of the tuples returned the inner block.
C(all but COUNT)	—
$\underline{C}$(all but COUNT)	Describe the nested predicate in a separate text.
$A, \underline{A}$	Use the same algorithm as SPJ queries, except that an aggregation noun should be used instead of a selection constant.

Table 1: Simple Nesting Types and Their Translation Strategy

Table 1 gives a summary of the handling of the above nesting types.

```
SELECT  S#
FROM    SUPPLY
WHERE   P# = '001'
  AND   QTY > ( SELECT  AVG(QTY)
                FROM    SUPPLY
                WHERE   P# = '001' )
```

Figure 4: An Example of a Type-A Nested Query

Example 4 : For the Type-A nested query shown in Figure 4, the modified version of Algorithm 1 gives the following pseudonatural language expression:

List [S#] such that:
supplier with supplier-code [S#] supplies parts with part-code '001' in quantity QTY (greater than the average of quantities for parts with part-code '001').

4.2 Single Nested Predicates and Natural Quantifiers

Single-level nested predicates which handle a value with respect to a set or which check the size of a set are useful for expressing such requests as: *"Who supplies at least one type of red parts?"* or *"List the parts supplied by no suppliers in London."*

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. We have to cope with the problem of such quantifier scopings, since any ambiguity in pseudonatural language translation may be hazardous rather than helpful for naive users.

The cause of such an ambiguity is that there is no clear distinction between the prerequisite of the domain of the quantified variable and the restriction to be satisfied on the domain. One solution, suggested in [12], is as follows:

For ≪*natural_quantifier*≫ ≪*variable*≫ which satisfy
≪ *domain_specification* ≫,
≪ *restriction_on_the_domain* ≫. (14)

The first interpretation of the previous example should be:

For every supplier, a red part is supplied by the supplier.

The other interpretation is:

For a red part, the part is supplied by every supplier.

We can adopt this format for our pseudonatural language text generation by preparing some more natural language fragments for plural forms of noun phrases. The general format of a quantified pseudonatural language expression is:

For ≪*quantifier*≫, ≪*quantified_noun*≫ ≪*subclause_for_domain_specification*≫,
≪*restriction_on_the_domain*≫. (15)

The previous algorithm should be slightly modified so that it can choose the head noun of the quantified attribute instead of the base sentence. The other phases of the algorithm (except the formatting phase) can then be used for generating the domain specification in Template (15). The restriction can be generated easily.

Category-Q, -M and -E nestings in SQL can be handled with quantified pseudonatural language expressions. The nesting types related to natural quantifiers are shown in Table 2.

(a) Modifications to Algorithm 1 for Category-Q Nestings

A Category-Q nesting with ANY or SOME quantifier checks whether or not there exists one value, in the set returned by the inner block, which satisfies the scalar comparison θ with the attribute value in the outer block. This can be described using *"at least one"* as the quantifier in Template (15). The domain specification is obtained from the inner block by the modified version of the algorithm in Section 3.

A Type-Q nesting with ALL returns TRUE if the attribute value satisfies the θ relationship for all the values returned by the inner block. This can be expressed using *"all"* as the quantifier.

Type	Natural Quantifier	Quantified Noun
Q, $\underline{Q}$	(ALL) "all" (ANY, SOME) *"at least one"*	The attribute returned by the inner block
C (COUNT)	—	—
$\underline{C}$ (=) (>) (≧) (<) (≦) (≠)	*"exactly <n>"* *"less than <n>"* *"at most <n>"* *"more than <n>"* *"more than <n>"* *"more or less than <n>"*	For COUNT(*), The inter block reference attribute; For COUNT(≪*attribute*≫)), ≪*attribute*≫;
M, $\underline{M}$	*"at least one"*	The attribute returned by the inner block
M^-, $\underline{M}^-$	*"no"*	The attribute returned by the inner block
E, E^-	—	—
$\underline{E}$	*"at least one"*	The inter block reference attribute
$\underline{E}^-$	*"no"*	The inter block reference attribute

Table 2: Nesting Types Associated to Natural Quantifiers

(b) Modifications to Algorithm 1 for Category-E Nestings

Type-$\underline{E}$ and Type-$\underline{E}^-$ nestings can be translated into quantified pseudonatural expressions with quantifiers *"at least one"* and *"no"*, respectively. The domain specification is in the inner block, and the restriction is in the outer block.

(c) Modifications to Algorithm 1 for Category-C Nestings with the COUNT Function

A Category-C predicate using the COUNT function also expresses the natural quantifier. In this case, ≪*constant*≫ must be a cardinal number. Let θ be '$\leq$' and the number be $n(> 0)$. Then, the predicate checks whether the interblock reference attribute has *"at least n"* values associated with it.

```
SELECT  S#
FROM    SUPPLY
WHERE   P#  IN  (  SELECT  P#
                   FROM    SUPPLY
                   WHERE   S# = '002'  )
```

Figure 5: An Example of a Type-M Nested Query

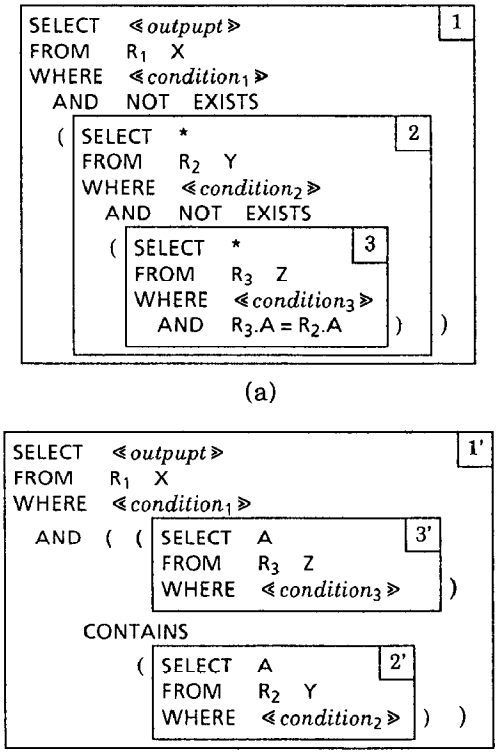
Example 5 : Suppose that we have a Type-M nested query in Figure 5. The domain specification is expressed in the inner block. The modified algorithm attaches the subclause for the

domain specification just after the phrase *"For at least one type of part "*. The restriction part, in the outer block, is translated into a sentence in the same manner as an SPJ query, except that the quantified attribute noun must be replaced with its plural.

List [S#] such that:
For at least one type of parts (those supplied by supplier with supplier-code '002'),
supplier with supplier-code [S#] supplies such parts.

4.3 Composite Nested Predicates in SQL

The inner block of a nested predicate may contain another block, or it may contain a conjunction or disjunction of several nested predicates. Such composite nested predicates are used to express more complex quantifications or to describe a combination of several quantified expressions.



(a)

(b)

Figure 6: Two Equivalent Nested Queries

Example 6 : The SQL query previously shown in Figure 2 contains a composite nesting. It is not easy to interpret such a complex query. In [10], however, the equivalence between the two nested queries shown in Figure 6 is proved. The query in Figure 6(b) uses a now-abolished operator CONTAINS. The meaning of the operation is more obvious in this formulation than in the original one. From this equivalence, we can now interpret the meaning of the query in Figure 2. This form of nested predicates are used for queries involving the quantifier *"all"*. The quantified pseudonatural expression of the query in Figure 2 is:

List [S#] such that:
For all types of parts (those which have color 'Red'),
supplier with supplier-code [S#] supplies those parts.

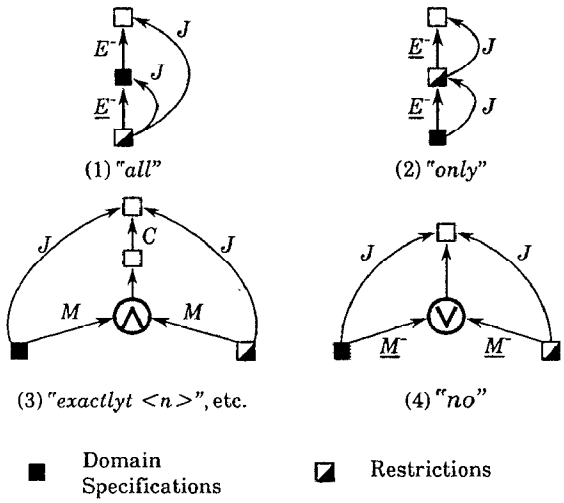


Figure 7: Composite Nested Predicates and Natural Quantifiers

We have analyzed various composite nested predicates and obtained nesting forms which can be translated into a quantified pseudonatural language expression. Figure 7 shows some of the cases for which we have obtained results so far. Shaded squares and half shaded squares indicate the domain specification and the restrictions, respectively.

Suppose that we have a nesting graph obtained from a given SQL query containing a composite nested predicate. If the graph is equivalent to one of those shown in Figure 7, we can obtain a pseudonatural language expression using the method discussed in the previous subsection.

Some composite nested predicates may have a structure similar (but not equivalent) to one of those in Figure 7. The following heuristics find such a similarity, if any. If the transformed graph matches one of those in Figure 7, our algorithms for text generation can generate a compact expression. Otherwise, we should translate each nested predicate separately.

Heuristic 1 (Subgraph Contraction) If there is a subgraph which is connected to only one block node directly or through an AND-node, collapse that subgraph into the node. If the whole graph matches any of the forms shown in Figure 7, use the modified algorithm for text generation. This will cause a nesting of quantified pseudonatural language expressions, if the collapsed subgraph expresses quantification.

Heuristic 1 searches a subgraph which is essentially a part of the domain specification or the restriction of the upper level quantification. If the subgraph itself corresponds to a quantified ex-

pression, its pseudonatural language translation is embedded in the domain specification or the restriction of the upper level.

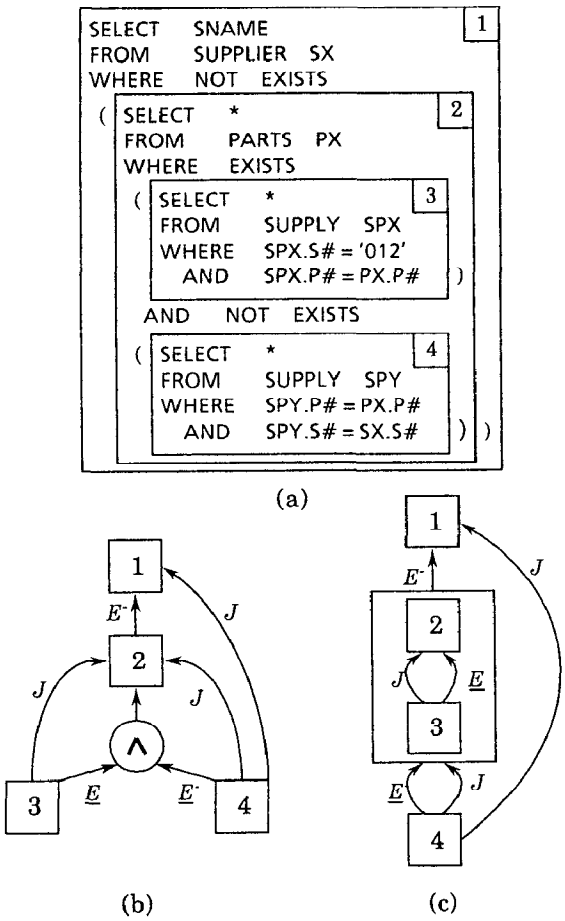


Figure 8: An Example of an Application of Heuristic 1

Example 7 : For an SQL query in Figure 8(a), Heuristic 1 collapses Block 3 into Block 2. We obtain the following pseudonatural language expression:

List [SNAME] such that:
For all types of parts (those which satisfy: For at least one of the same type of parts, supplier with supplier-code '012' supplies those parts.),
supplier with supplier-code {S#} (called [SNAME]) supplies those parts.

Heuristic 2 (Branch Split) If there is an OR node which has no interblock reference between a block beneath it and a block above it, split up the branches below the OR node by copying its outer block node and raising the OR node. Repeat this until:
- The OR node jumps over the outermost block node;
or,
- The outer block node of the OR node is adjacent to a Category-*A* nesting edge.

Heuristic 2 splits up OR-branches, and generates several pseudonatural language expressions to be combined with "or" in the output text. Aggregations, however, prohibit this transformation, since we cannot in general calculate an aggregated value for a set on its subsets. This heuristic is based on the observation that a pseudonatural language expression is conjunction-oriented. Since it assumes a conjunction when there are several modifiers in one sentence, the occurrence of "or" in the middle of the expression may obscure its meaning.

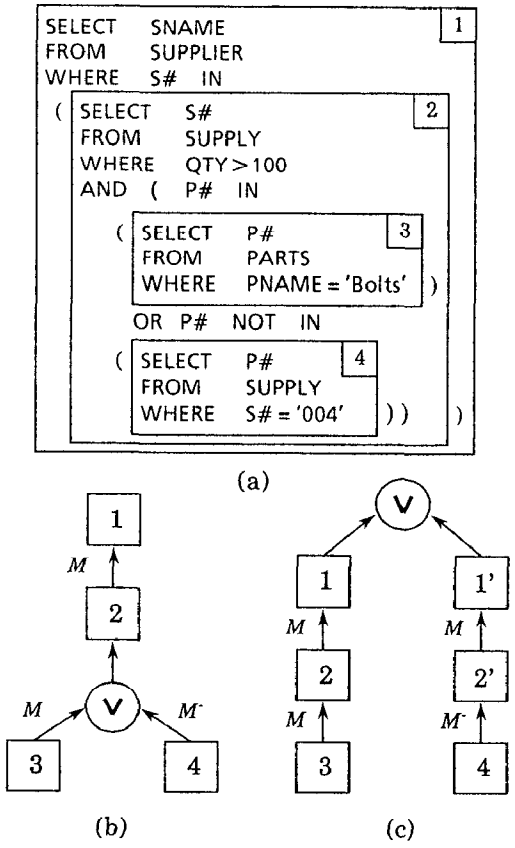


Figure 9: An Example of an Application of Heuristic 2

Example 8 : For an SQL query in Figure 9(a), Heuristic 2 transforms the nesting graph shown in Figure 9(b) into the one in (c). We obtain the following pseudonatural language expression:

List [SNAME] satisfying (a) or (b):

(a) For at least one supplier with supplier-code {S#} (those which satisfy:
For at least one type of parts (those which are called 'Bolts'),
supplier with supplier-code {S#} supplies those parts in quantity {QTY} (more than 100)),
the supplier with supplier-code {S#} are called [SNAME].

(b) For at least one supplier with supplier-code {S#}
(those which satisfy:
For no types of parts (those which are supplied
by supplier with supplier-code '004'),
supplier with supplier-code {S#} supplies
those parts in quantity (more than 100)),
the supplier with supplier-code {S#} are called
[SNAME].

5 Concluding Remarks

In this paper, we discussed how to translate SQL queries into pseudonatural language expressions. This will reduce burdens on novice users who want to retrieve information satisfying complex specifications. This approach employs no complex natural language processing technique. Localized information for individual application can be prepared by a database administrator or the user in the case of personal databases. SQL queries analyzed in this paper contain both simple and composite nested predicates difficult to understand. Our method will help novice users to learn and use the more difficult features of SQL. It can be used for SQL tutoring systems.

Pseudonatural language has other possible applications. First, it can be used for translating *stored queries*. As the needs and uses of databases grow, an increasing number of queries will be written and executed. Those queries themselves may be too precious to be abandoned after only one execution. If a query database facility[5] is established, users can compose a new query from components stored in a database. The readability problem of these stored queries can be solved by our method.

Furthermore, it can be used in a cooperative environment. When a user works with others, he or she must understand their works. A pseudonatural language translator will help him or her to understand queries written by others. Since the translator will add application specific information to these queries, the user will find it easier to relate these queries with the work at hand.

Acknowledgments

The authors would like to thank Mr. Mohamed E. El-Sharkawi and Mr. Sunao Sawada for their discussions and collaborations. They are also grateful to Dr. Michael E. Houle who gave valuable comments concerning this paper.

This research was supported in part by the Ministry of Education, Science, and Culture of the Government of Japan, under a Grant-in-Aid for Co-operative Research (Grant-No. 01302059).

References

[1] Berge, C.: *"Graphs and Hypergraphs,"* North-Holland, 1976.

[2] Date, C. J.: *"A Guide to the SQL Standard,"* Addison-Wesley, 1987.

[3] El-Sharkawi, M. E. and Kambayashi, Y.: "The Architecture and Implementation of ENLI: Example-Based Natural Language-Assisted Interface, " *Proc. PARBASE-90: Int. Conf. on Databases, Parallel Architectures, and Their Applications,* pp. 430–432, March 1990.

[4] Epstein, S. S.: "Transportable Natural Language Processing through Simplicity — The PRE System," *ACM Trans. Office Inf. Syst.*, Vol. 3, No. 2, pp. 107–120, April 1986.

[5] Kambayashi, Y.: "Functions of Database Workbench," *Proc. AFIPS National Comput. Conf.,* Vol. 53, pp. 547–553, July 1984.

[6] Kambayashi, Y.: "An Overview of Natural Language-Assisted Database User Interface: ENLI," *Proc. IFIP 10th World Comput. Cong.*, pp. 1055–1060, September 1986.

[7] Kambayashi, Y. and Amano, H.: "Transformations of Natural Language Expressions by Basic Relational Database Operations," *Trans. IPS Japan*, Vol. 30, No. 10, pp. 1316–1323, October 1989 (in Japanese).

[8] Kim, W.: "On Optimizing an SQL-Like Nested Query," *ACM Trans. Database Syst.,* Vol. 7, No. 3, pp. 443–469, September 1982.

[9] Lowden, B. G. T. and De Roeck, A. N.: "The REMIT System for Paraphrasing Relational Query Expressions into Natural Language," *Proc. 12th Int. Conf. on Very Large Data Bases*, pp. 365–371, August 1986.

[10] Luk, W. S. and Kloster S.: "ELFS: English Language from SQL," *ACM Trans. Database Syst.*, Vol. 11, No. 4, pp. 447–472, December 1986.

[11] Thomas, J. C.and Gould, J. D.: "A Psychological Study of Query by Example," *Proc. AFIPS National Comput. Conf.*, pp. 439–445, May 1975.

[12] Woods, W. A.: "Semantics and Quantification in Natural Language Question Answering," *Advances in Computers*, Vol. 17, pp. 1–87, Academic Press, 1978.

[13] Zloof, M. M.: "Query by Example," *Proc. AFIPS National Comput. Conf.*, pp. 19–22, May 1975.