On mapping natural language constructs into relational algebra through E-R representation[*]

FRANK S.C. TSENG
Department of Computer Science and
Information Engineering
National Chiao Tung University
Hsinchu, Taiwan, ROC

ARBEE L.P. CHEN
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan, ROC
Fax: 886-35-723694

WEI-PANG YANG
Department of Computer Science and
Information Engineering
National Chiao Tung University
Hsinchu, Taiwan, ROC

Abstract

Research on accessing databases in natural language usually employs an intermediate form for the mapping process from natural language to database languages. However, much effort is needed to bridge the gap between the existing intermediate forms and the database languages. In this paper, we present a methodology to map natural language constructs into relational algebra through E-R representation. This methodology employs a logical form to represent the natural language queries. The logical form has the merits that it can be mapped from natural language constructs by referring to the Entity-Relationship conceptual schema and can be efficiently transformed into relational algebra for query execution. The whole process provides a clear and natural framework for processing natural language queries to retrieve data from database systems.

keywords: relational databases, conceptual schema, natural language queries, Entity-Relationship model, semantic role, query transformation, relational algebra, surrogate, logical form.

[*]This research was partially supported by the Republic of China National Science Council under contract No. NSC 81-0408-E-007-12.

Contents

List of Figures

1 Introduction

In database management systems, data are retrieved via a well-defined query language. Although some query languages, say SQL [9] (Structural Query Language), can be very powerful, users may suffer greatly from their complex usage. For example, based on the Suppliers-and-Parts database [10] (see Figure 3), the query "Get supplier names who supply red parts." can be issued as follows (in SQL).

```
SELECT sname
FROM   Suppliers, Parts, Shipments
WHERE  Shipments.sno = Suppliers.sno AND
       Shipments.pno = Parts.pno     AND
       Parts.color = 'red';
```

Users have to know the structure of the underlying schema in detail. First of all, they must determine which relations will be needed, then which attributes are involved, and finally how to join these relations.

Moreover, a formal query may become very complex even when the corresponding natural language query has changed just a little bit. For instance, if the previous example is changed into "Get supplier names who supply *all* red parts." Then, the corresponding SQL query becomes (refer to [10, Section 6.5.9])

```
SELECT sname
FROM   Suppliers
WHERE  NOT EXISTS
       (SELECT *
        FROM   Parts
        WHERE  NOT EXISTS
               (SELECT *
                FROM   Shipments
                WHERE  Shipments.sno = Suppliers.sno AND
                       Shipments.pno = Parts.pno     AND
                       Parts.color = 'red'));
```

It is not only hard to understand this formal query, but also more complex than the previous one. The complexity of formal query languages thus usually frustrates naive users.

Research in natural language processing for DBMS interface attempts to ease this complexity by freeing users from knowing the exact database structure and learning the query language. Natural language query systems are more feasible than general purpose language processors, since the number of relevant objects to be described in the semantic knowledge base is naturally restricted in its application domains. However, as pointed out in [21], a frequent criticism concern of natural language interfaces is that we cannot expect natural language interfaces to act appropriately for every input sentence. Therefore, we hope to limit our work in processing the following types of query sentences.

- *Interrogative* — “Does Smith supply nuts?”
- *Imperative* — “List all the suppliers.”
- *Declarative* — “Smith supplies nuts.” (Which is treated as a question.)

Moreover, users should be aware that the system might be unable to provide an answer if their expectations exceed the actual database capability, since the information stored in a database is just a precise world subset. If the system cannot make a decision to get answers due to some ambiguity, than users are asked to answer some questions to clarify the ambiguity.

1.1 Objectives

An intermediate form is usually employed for mapping natural language constructs into the underlying database schema. This intermediate form can be a universal relation [19, 20] as used in FRED [12, 16]. It can also be a hierarchy structure constructed from the information about the database. TEAM [11] and QPROC [22] are two example systems of this type. Besides, the TQA [7] system use a semantic net model as its intermediate representation.

However, Ullman has pointed out that interpreting queries over a universal relation is a difficult task [20]. Other intermediate forms suffer from the bias to natural language constructs and much effort is needed to translate the intermediate forms into the database languages.

In general, queries to a database can be mapped into a subset of its conceptual schemas. We recommend to organize the conceptual schema by Entity-Relationship (E-R) model [3] for constructing a natural language interface. In this paper, we study the inter-relationship between natural language constructs and the conceptual schema as shown in the shadow area in Figure 1. Chen [4] has pointed out that the basic constructs of English sentences

Figure 1: The Focus in This Paper is on the Shadow Area.

can be mapped into E-R schemas in a natural way. Chen [4] has studied 11 rules for translating information requirements, which are originally documented in English, into database schemas in terms of E-R diagrams. In comparison, we propose an approach to map the natural language queries into relational algebra through the E-R representation.

1.2 Review of E-R Model

The Entity-Relationship model [3] adopts the view that the real world consists of *entities* and *relationships* among entities. An entity is a 'thing' which can be distinctly identified.

A specific person, company or event is an example of an entity. A relationship is an association among entities. The E-R model uses the concepts of *entity set, relationship set* and *value set.* An entity set represents the generic structure of an entity in an enterprise's realm of interest. A relationship set represents the generic structure of a relationship among entity sets. A value set is a domain of either an entity set or a relationship set.

The structure of a database organized according to the E-R data model can be depicted by an Entity-Relationship Diagram (ERD). In an ERD, an entity set is represented by a rectangular, and a relationship set is represented by a diamond, labeled by their associated names. The entity sets that participate in a relationship set are indicated by edges with their corresponding semantic roles attached. Moreover, a value set is represented as an oval labeled with the attribute defined on it. Figure 2 depicts the ERD of the Suppliers-and-Parts database described in [10].

Figure 2: The E-R Diagram for the Suppliers-and-Parts Database.

Information about an entity set can be organized into a relation which is named *entity relation.* Similarly, a *relationship relation* can be constructed by collecting the attributes of the relationship set with the primary keys of the associated entity relations. For example, in Figure 3, the relations Suppliers and Parts of the Suppliers-and-Parts database shown in Figure 2 are *entity relations*, while Shipments is a *relationship relation.* Our examples will be based on this database hereafter.

Suppliers

sno	*sname*	*status*	*city*
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adames	30	Taipei

Parts

pno	*pname*	*color*	*weight*	*city*
P1	Nuts	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

Shipments

sno	*pno*	*qty*
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

Figure 3: The Suppliers-and-Parts Database.

1.3 Overview of Natural Language Processing

Language understanding process is commonly divided into three stages. First, the sentence is parsed according to the predefined grammar, then the semantic roles are built and finally these semantic roles are mapped into the specific objects in the real world.

Augmented-Transition-Net (ATN) [24] is traditionally used to parse a natural language sentence. By following a method for expressing grammars in logic due to Kowalski [13], Pereira and Warren [15] have developed a clearer and more powerful formalism named Definite Clause Grammar (DCG). McCord [14] contributes to the syntactic analysis and semantic interpretation of natural languages in the framework of logic programming.

Winston [23] described a variety of constraints to help establish semantic roles in a sentence. These semantic roles reveal how the nouns are related to the verb. Consider the sentence "Andy moves the dirty table into the messy office." It is parsed into a noun phrase and a verb phrase. The verb phrase consists of a noun phrase and a prepositional phrase which consists of another noun phrase.

The semantic roles of the sentence are shown in Figure 4(a). It indicates that the *verb* is move, the *subject* is Andy, the *object* is the dirty table, and the *destination* is the messy office. Finally, these semantic roles can be mapped into specific objects in the real world as Figure 4(b) illustrates.

Figure 4: (a) The Semantic Roles for “Andy moves the dirty table into the messy office.”
(b) Mapping Semantic Roles into Specific Objects.

Our approach follows these three stages and we focus on the mapping from semantic roles to an E-R schema. We develop a logical form to represent the mapping result and it can be efficiently transformed into the relational algebra for query execution.

The remainder of this paper is organized as follows. Section 2 describes the processing model and the mapping process. Section 3 devotes to the logical form constructs which can be used to represent the mapping result of a user query. Query transformation process which transforms logical forms into the relational algebra is described in Section 4. Finally, we conclude and suggest future research in Section 5.

2 The Processing Model and the Mapping Process

2.1 The Processing Model

The processing model is shown in Figure 5. There is a front-end parser/mapper to parse the English queries and map them into the underlying E-R schema. The parsing and mapping process may refer to

1. the dictionary and

2. the underlying E-R schema associated with the semantic-role frame (ER/SRF).

After the parsing phase, the query is decomposed into

Figure 5: The Processing Model.

1. *semantic roles*, each of which is composed of a *headnoun* and some *modifiers* (a headnoun is the main noun in a noun phrase; for example, "the London suppliers" has a headnoun 'suppliers' and the other noun 'London' is a modifier which modifies the headnoun) and

2. the *verb* that relates these semantic roles.

Each of the semantic roles is mapped into an *entity relation* and its headnoun and modifiers are mapped into the corresponding attributes of that entity relation based on ER/SRF (to be discussed in Section 2.1.2). The verb that relates these semantic roles is mapped into the relationship relation that associates these entity relations. Figure 6 illustrates this mapping mechanism.

We develop a logical form in Section 3 to represent the mapping result. After the logical form is generated, it is passed to the query transformer to produce the query in relational algebra.

Our processing model makes the following assumptions.

1. In addition to the original attributes, each entity relation is augmented with a *surrogate*. Any reference to this surrogate is interpreted as a reference to the corresponding relation. For example, in the Suppliers-and-Parts database, we add the extra attributes *sno* and *pno* as the surrogates of Suppliers and Parts, respectively.

2. We also augment each relationship relation with a surrogate which is composed of the surrogates of the involved entity relations. For example, the surrogate of the relationship relation Shipments is formed by grouping *sno* and *pno* into *(sno, pno)*.

Figure 6: The Mapping Mechanism.

2.1.1 The Dictionary

The linguistic knowledge that enables the semantic roles of a verb to be mapped into the correct attributes of the associated relationship relation is stored in the dictionary. Notice that some of these attributes are surrogates of the entity relations involved in the relationship. For the Suppliers-and-Parts database, the knowledge for the verb 'supply' would be like

Relation	Semantic Role	the Corresponding Attribute
Shipments	Subject	sno (the surrogate of Suppliers)
	Object	pno (the surrogate of Parts)
	with_object	qty

Other information such as the synonyms of the entity sets with the corresponding relations and all the domain values with the corresponding attributes are also needed to be stored in the dictionary.

2.1.2 The Schema and the Semantic-Role Frame Represented in ERD

For each entity relation, attributes are identified as the *headnoun* and the *modifiers* that modify the headnoun. For example, the attributes corresponding to the headnouns of Suppliers and Parts are *sname* and *pname*, respectively. Other attributes are identified to be the corresponding modifiers. The above information will be encoded into the original ERD. For example, the schema and the semantic-role frame representing the Suppliers-and-Parts database are shown in Figure 7.

Figure 7: The Schema and the Semantic-Role Frame of the Suppliers-and-Parts Database.

2.2 Description of the Mapping Process

2.2.1 Mapping Verbs into Relationship Relations

By referring to the dictionary, we can map the verb of a query sentence into the corresponding relationship relation and its semantic roles into the corresponding attributes. Besides, an adverb phrase that modifies the verb can be mapped into the corresponding attribute of the relationship relation. The query "Does Smith supply nuts with quantity 300?", for instance, has the semantic roles which can be mapped into the attributes as depicted in Figure 8. The attributes *sno* and *pno* refer to the entity relations Suppliers

Figure 8: The Mapping for "Does Smith supply nuts with quantity 300?"

and Parts, respectively. That tells us 'Smith' and 'nuts' will be associated with the headnouns of the entity relations Suppliers and Parts, respectively. Moreover, '300' will be mapped into the attribute *qty* of the relationship relation Shipments.

Note that the verb 'be' is treated differently, for it is a linking verb, which is followed by a subject complement which describes the subject. Linking verbs do not transfer action; rather, they join the subject and the complement. For example, consider

"Smith is a London supplier."

in which *supplier* describes *Smith*. Because the verb 'be' does not transfer action, we need not associate it with a *relationship relation*. The above sentence is mapped into the entity relation Suppliers (and the attributes *sname* and *city*, see Section 2.2.2).

Note also that an imperative mood sentence is always used for issuing command. The leading verb does not transfer action. For example,

"List the suppliers located in London."

is mapped into the entity relation Suppliers (and attribute *city*).

2.2.2 Mapping Noun Phrases into Entity Relations and Its Modifiers into Attributes

As we have shown in the previous section, the semantic roles of a verb in a sentence can be mapped into those surrogates of involved entity relations. Those entity relations actually interpret noun phrases of the sentence.

However, the noun phrases may consist of more than one noun modifiers (which can be adjectives or nouns) or relative clauses that modify the headnoun. For example, "the London suppliers" and "the red parts" are two noun phrases and have the noun modifiers 'London' and 'red', respectively. These noun modifiers can be interpreted into the attributes of the corresponding entity relation. According to the dictionary, the interpretation of these noun modifiers described above can be mapped as Figure 9 illustrates.

Notice that some relationships can be queried in possessive form like "List A's B." (or "List the B of A.") In this case, there is no verb to be mapped into the relationship relation. But, by referring to the schema and the semantic-role frame represented in ERD,

Noun Phrase	Corresponding Relation	Interpretation of the Noun Modifier
the London suppliers	Suppliers	city = ‘London’
the red parts	Parts	color = ‘red’

Figure 9: Mapping Noun Modifiers into Attributes.

we may first map the noun phrases (A and B) into the corresponding entity relations and then establish the path between the entity relations according to the ERD. This path is obtained as the mapping result. For example, if we have the following schema then the

query “List Arbee’s student.” can be mapped as follows.

Then the path ($\boxed{\text{Teacher}}$<Teach>$\boxed{\text{Course}}$<Take>$\boxed{\text{Student}}$) is our mapping result, where relationship relations are enclosed by angle brackets. Note that this query is equivalent to “List the students who take the courses taught by Arbee.”

Nevertheless, for a general schema, there may exist many paths between two entity relations in the ERD. In such circumstances, users are asked to choose one of the relationships, since it is inherently ambiguous, even a human being cannot distinguish it. For instance, if there exists another <Advise> relationship between $\boxed{\text{Teacher}}$ and $\boxed{\text{Student}}$, then “List Arbee’s student.” will cause two ambiguous interpretations. One is

“List the students who take the courses taught by Arbee.”

and the other is

“List the students who are advised by Arbee.”

3 The Logical Form

In this section, we develop a logical form for representing the results of mapping from English queries into an E-R schema. It can be easily translated into the relational algebra for query execution.

3.1 The Extension of E-R Diagram for the Logical Form

Generally speaking, queries to a database can be constructed from a subset of its conceptual schema [21]. Therefore, based on the constructs of E-R diagram, we develop a logical form to represent natural language queries by extending the constructs of E-R diagram, which represents the underlying schema. The logical form takes into account the following conditions of an English query.

1. The representation for modifiers,
2. The representation for conjunctives 'and' and 'or', and
3. The representation for the word 'all'.
4. The representation for the negative and affirmative forms of a verb,

These conditions are explained in the following subsections.

3.1.1 The Representation for Modifiers

After the headnoun and modifiers of an entity set are recognized, we respectively associate them with the corresponding attributes and form them into predicates of the form "attribute θ constant", $\theta \in \{>, <, =, \neq, \geq, \leq\}$. These predicates are represented by oval nodes in our logical form. For example, "List the suppliers who supply red parts." has the logical form shown in Figure 10. We define the predicate "attribute = ?" as a *pseudo predicate*; it represents the target attribute which is to be output to the user.

3.1.2 The Representation for Conjunctives 'And' and 'Or'

The conjunctive 'and' is sometimes interpreted to be a disjunction (logical 'or'). For example, "List the red parts and the blue parts." in logical sense is equivalent to "List the red parts or the blue parts." We will explore such phenomenon in the following.

Figure 10: The Logical Form of "List the suppliers who supply red parts."

From the point of view of E-R model, 'and' and 'or' can be used to conjunct the following three cases.

1. *Modifier and Modifier.* In this case, the 'and' acts exactly as logical 'and'. The oval nodes that correspond to these modifiers are linked pairwise by edges labeled '$\wedge$' and '$\vee$' for conjunctives 'and' and 'or', respectively. For example, "List the parts *with color blue* and (or) *located in London.*" has the logical form shown in Figure 11(a).

Figure 11: (a) The Logical Form for "List the parts with color blue and (or) located in London." (b) The Logical Form for "List the suppliers who supply red parts and (or) nuts." (c) The Logical Form for "List the suppliers who supply red parts and (or) supply nuts."

2. *Entity Set and Entity Set.* In this case, a conjunctive conjuncts two noun phrases. The rectangle nodes that correspond to the nouns are linked pairwise by edges labeled '$\wedge$' and '$\vee$' for conjunctives 'and' and 'or', respectively. For instance, "List the suppliers who *supply red parts* and (or) *nuts.*" has the logical form shown in Figure 11(b). However, when the rectangle nodes linked by an $\wedge$-edge are respectively connected to a pseudo predicate, then the $\wedge$-edge must be changed into an $\vee$-edge. That is, the 'and' that conjuncts these two noun phrases is actually a logical 'or', since it is used here to conjunct two noun phrases which are not related by any relationship. The 'and' means that the user wants to obtain *both* the answer represented by the two pseudo predicates and want to combine them together. For example, the logical form of "List the red parts and the blue parts." is

3. *Relationship and Relationship.* In this case, a conjunctive conjuncts two verb phrases and the 'and' acts exactly as the logical 'and'. The diamond nodes that correspond to the verbs are linked pairwise by edges labeled '$\wedge$' and '$\vee$' for conjunctives 'and' and 'or', respectively. For instance, "List the suppliers who *supply red parts* and (or) *supply nuts.*" has the logical form shown in Figure 11(c). Note that the following example does not fall into this case.

   List the suppliers who supply nuts and are located in London.

   Although the 'and' conjuncts two verbs, the verb "are located in" is used to modify the suppliers instead of performing action. This sentence is equivalent to "List the London suppliers who supply nuts." and there is no conjunction at all.

Finally, there are problems of scoping with 'and' and 'or' when they are appeared together. Our representation for dealing this problem is analogous to using parentheses. For example, to distinguish $((A \wedge B) \vee C)$ from $(A \wedge (B \vee C))$, the logical form in Figure 12(a) is used for the former and that in Figure 12(b) is used for the later.

Figure 12: (a) The Logical Form for $((A \wedge B) \vee C)$. (b) The Logical Form for $(A \wedge (B \vee C))$.

3.1.3 The Representation for the Word 'All'

Consider the following examples,

1. List the suppliers who supply *all* red parts.

2. List the suppliers who supply red parts.

The answer of the first example is the suppliers who supply *all* the red parts, but the second example is the suppliers who supply *any* red parts. Therefore, the answer of (1) is always contained in that of (2). We use a shadow rectangle to represent the entity set whose semantic role is preceded by 'all'. Otherwise, a blank rectangle is used (see Figure 13). Note that the semantic meaning of "List *all* the suppliers who supply *all* red parts." is equivalent to that of (1). The first 'all' has no effect on this query.

3.1.4 The Representation for the Negative and Affirmative Forms of a Verb

A verb is usually mapped into a relationship relation, which is depicted as a diamond in the E-R diagram. But the verb in a query may be issued in negative form, which negates the relationship to which the verb corresponds. For example, a user may issue a query in negative form like "List the suppliers who *do not supply* nuts." We extend the diamond

Figure 13: (a) The Logical Form for "List the suppliers who supply all red parts."
(b) The Logical Form for "List the suppliers who supply red parts."

representation of E-R diagram to represent both the negative and affirmative form of a query as Figure 14 shows. In affirmative case, we represent the relationship as before;

Figure 14: (a) The Relationship Representation in Affirmative Form.
(b) The Relationship Representation in Negative Form.

in negative case, we represent the relationship as a diamond except that there is a black triangle that links to the entity set whose semantic role *does not satisfy* the relationship. In the previous query, the black triangle links to the entity set Suppliers.

Notice that, together with the representation for the word 'all', we can obtain eight cases as depicted in Figure 15 that enumerates the effects of the combinations of *negation/affirmation* and *universal/existential quantifier* to a relationship. In Figure 15, R represents a relationship and S and O represent its subject and object, respectively.

No.	*logical form*	*interpretation of answer*
1		$\{x \mid (\exists s)(\exists o)((s \in S \wedge o \in O \wedge R(s,o)) \Rightarrow x = s.A)\}$
	e.g. “List the A of S who supply O.”	
2		$\{x \mid (\exists s)(\exists o)((s \in S \wedge o \in O \wedge \neg R(s,o)) \Rightarrow x = s.A)\}$
	e.g. “List the A of S who *do not supply* O.”	
3		$\{x \mid (\exists s)(\forall o)((s \in S \wedge o \in O \wedge R(s,o)) \Rightarrow x = s.A)\}$
	e.g. “List the A of S who supply *all* O.”	
4		$\{x \mid (\exists s)(\neg(\forall o)(s \in S \wedge o \in O \wedge R(s,o)) \Rightarrow x = s.A)\}$
	e.g. “List the A of S who *do not supply all* O.”	
5		$\{y \mid (\exists s)(\exists o)((s \in S \wedge o \in O \wedge R(s,o)) \Rightarrow y = o.A)\}$
	e.g. “List the A of O which are supplied by S.”	
6		$\{y \mid (\exists s)(\exists o)((s \in S \wedge o \in O \wedge \neg R(s,o)) \Rightarrow y = o.A)\}$
	e.g. “List the A of O which *are not supplied* by S.”	
7		$\{y \mid (\exists o)(\forall s)((s \in S \wedge o \in O \wedge R(s,o)) \Rightarrow y = o.A)\}$
	e.g. “List the A of O which are supplied by *all* S.”	
8		$\{y \mid (\exists o)(\neg(\forall s)(s \in S \wedge o \in O \wedge R(s,o)) \Rightarrow y = o.A)\}$
	e.g. “List the A of O which *are not supplied* by *all* S.”	

Figure 15: The combinations of negation/affirmation and universal/existential quantifier to a relationship.

3.2 Formal Definition for the Logical Form

From the previous discussion, a logical form for a query Q can be denoted by $LF(Q) = (N, E, f_N, f_E)$, where

1. N is a set of nodes which can be further classified into the sets N_r, N_e, and N_p (i.e. $N = N_r \cup N_e \cup N_p$), where
   (a) N_r is the set of diamond nodes representing *relationship relations* or their conjunction/disjunction,
   (b) N_e is the set of rectangle nodes representing *entity relations* or their conjunction/disjunction,
   (c) N_p is the set of oval nodes representing *predicates* which is of the form "attribute θ constant", $\theta \in \{>, <, =, \neq, \geq, \leq\}$, or their conjunction/disjunction.
2. E is a set of edges, which can be further classified into the sets $E_{(r,e)}$, $E_{(r,p)}$, $E_{(e,p)}$, $E_{(p,p)}$, $E_{(e,e)}$, and $E_{(r,r)}$ (i.e. $E = E_{(r,e)} \cup E_{(r,p)} \cup E_{(e,p)} \cup E_{(p,p)} \cup E_{(e,e)} \cup E_{(r,r)}$), where
   (a) $E_{(r,e)} \subseteq N_r \times N_e$. An edge $(r, e) \in E_{(r,e)}$ is said to join the diamond node r and the rectangle node e.
   (b) $E_{(r,p)} \subseteq N_r \times N_p$. An edge $(r, p) \in E_{(r,p)}$ is said to join the diamond node r and the oval node p.
   (c) $E_{(e,p)} \subseteq N_e \times N_p$. An edge $(e, p) \in E_{(e,p)}$ is said to join the rectangle node e and the oval node p.
   (d) $E_{(p,p)} \subseteq N_p \times N_p$. An edge $(p_1, p_2) \in E_{(p,p)}$ is said to join the oval nodes p_1 and p_2.
   (e) $E_{(e,e)} \subseteq N_e \times N_e$. An edge $(e_1, e_2) \in E_{(e,e)}$ is said to join the rectangle nodes e_1 and e_2.
   (f) $E_{(r,r)} \subseteq N_r \times N_r$. An edge $(r_1, r_2) \in E_{(r,r)}$ is said to join the diamond nodes r_1 and r_2.
3. f_N is a set of mappings, $f_N = \{f_{Nr}, f_{Ne}, f_{Np}\}$, where

(a) $f_{Nr} : N_r \rightarrow R_N$, where R_N is the set of the relationship relation names labeled on $r_i, \forall r_i \in N_r$.

(b) $f_{Ne} : N_e \rightarrow E_N \times \{\forall, \exists\}$, where E_N is the set of the entity relation names labeled on $e_i, \forall e_i \in N_e$. $\{\forall, \exists\}$ represents the cases 'all' ($\forall$) and 'any' ($\exists$) as addressed in Section 3.1.3.

(c) $f_{Np} : N_p \rightarrow P$, where P is the set of the predicates labeled on $p_i, \forall p_i \in N_p$.

4. f_E is a set of mappings, $f_E = \{f_{Ere}, f_{Erp}, f_{Eep}, f_{Epp}, f_{Eee}, f_{Err}\}$, where

   (a) $f_{Ere} : E_{(r,e)} \rightarrow S_r \times \{A, N\}$, where S_r is the set of the labels of $(r_i, e_j), \forall (r_i, e_j) \in E_{(r,e)}$, which represent the *semantic roles* of the rectangle nodes e_j. $\{A, N\}$ represents the affirmative (A) and negative (N) cases discussed in Section 3.1.4.

   (b) $f_{Erp} : E_{(r,p)} \rightarrow VM$, where VM is the set of labels of $(r_i, p_j), \forall (r_i, p_j) \in E_{(r,p)}$, which represent the *verb modifiers* of the verb corresponding to the diamond nodes r_i.

   (c) $f_{Eep} : E_{(e,p)} \rightarrow \{\text{headnoun, modifier}\}$.

   (d) $f_{Epp} : E_{(p,p)} \rightarrow \{\wedge, \vee\}$, where '$\wedge$' and '$\vee$' are the edge labels representing conjunction and disjunction, respectively.

   (e) $f_{Eee} : E_{(e,e)} \rightarrow \{\wedge, \vee\}$.

   (f) $f_{Err} : E_{(r,r)} \rightarrow \{\wedge, \vee\}$.

Figure 16: The Logical Form for Example 3.1.

EXAMPLE **3.1** For the query $Q =$ "List the London suppliers who *do not* supply *all* red parts.", the logical form $LF(Q) = (N, E, f_N, f_E)$ as represented in Figure 16 is formally specified as follows:

1. $N = \{r_1, e_1, e_2, p_1, p_2, p_3\} = N_r \cup N_e \cup N_p$, where $N_r = \{r_1\}, N_e = \{e_1, e_2\}$, and $N_p = \{p_1, p_2, p_3\}$.

2. $E = \{(r_1, e_1), (r_1, e_2), (e_1, p_1), (e_1, p_2), (e_2, p_3)\} = E_{(r,e)} \cup E_{(r,p)} \cup E_{(e,p)} \cup E_{(p,p)} \cup E_{(e,e)} \cup E_{(r,r)}$, where $E_{(r,e)} = \{(r_1, e_1), (r_1, e_2)\}, E_{(e,p)} = \{(e_1, p_1), (e_1, p_2), (e_2, p_3)\}$. $E_{(r,p)}, E_{(p,p)}, E_{(e,e)}$, and $E_{(r,r)}$ are all empty sets.

3. $f_N = \{f_{Nr}, f_{Ne}, f_{Np}\}$, where

   $f_{Nr}(r_1) = $ (Shipments)
   $f_{Ne}(e_1) = $ (Suppliers, $\exists$), $f_{Ne}(e_2) =$(Parts, $\forall$),
   $f_{Np}(p_1) =$ "$sname =?$", $f_{Np}(p_2) =$ "$city =$ London", and
   $f_{Np}(p_3) =$ "$color =$ red".

4. $f_E = \{f_{Ere}, f_{Eep}\}$, where

   $f_{Ere}((r_1, e_1)) = $ (Subject,N), $f_{Ere}((r_1, e_2)) = $ (Object,A),
   $f_{Eep}((e_1, p_1)) = $ 'headnoun',
   $f_{Eep}((e_1, p_2)) = $ 'modifier', and $f_{Eep}((e_2, p_3)) = $ 'modifier'. $\square$

Note that this definition of the logical form can be extended if more natural language constructs are to be taken into account in the mapping process. For example, we may add the constructs for aggregation functions (i.e. Max, Min, Avg, Sum, and Count).

4 The Transformation of a Logical Form into Relational Algebra

We select relational algebra as our target for the following reasons.

1. For the consideration of query optimization.

2. The relational algebra can be further transformed into other query languages (e.g. SQL or QUEL [8]) either for portability consideration [16, 18] or for distributed database retrieval [5, 6]. (For example, the transformation from relational algebra into SQL can be found in [10, Exercise 13.4])

Recall that a *pseudo predicate* is a predicate of the form "attribute = ?". The *target attributes* of a query Q, denoted $T(Q)$, are defined as the set of the attributes involved in all the pseudo predicates of the logical form $LF(Q)$. The *i-th component* of an n-tuple $t = (c_1, c_2, \ldots, c_n)$ is denoted $\pi_i(t) \equiv c_i$. In Example 3.1, $\pi_1(f_{Ne}(e_1)) =$ 'Suppliers' and $\pi_2(f_{Ne}(e_1)) =$ '$\exists$'.

In the following, the notations used in [19] will be adopted, in which $\sigma, \pi, \div, \bowtie, \cup, \cap, -$, and $\ltimes$ represent selection, projection, division, join, union, intersection, difference, and semijoin, respectively. The transformation process is discussed based on whether the logical form contains diamonds node or not. In Section 4.1, we discuss the case where the logical form contains no diamond nodes. In Section 4.2, we devote to the cases where the logical form contains one or more diamond nodes.

4.1 The Transformation Process for a Logical Form Containing No Diamond Nodes

In such cases, if there are edges of $E_{(e,e)}$, say (e_i, e_j), then e_i and e_j can be separately transformed by the following steps and the result is their union/intersection depending on (e_i, e_j) is an $\vee$-edge or $\wedge$-edge, respectively.

1. Use the predicates, except for pseudo predicates, to restrict the entity relation.

2. Project $T(Q)$.

That is, $LF(Q)$ will be transformed into $\pi_{T(Q)}(\sigma_P(\pi_1(f_{Ne}(e))))$, where e is the single rectangle node, and P is the compound predicate composed by the following function.

FUNCTION **4.1** *compose_predicate(LF)*

Input: a logical form, $LF = (N_p, E_{(p,p)}, f_{Np}, f_{Epp})$, containing only oval nodes linked by $\wedge$-edges and/or $\vee$-edges.

Output: the compound predicate P.

```
1. if (there is an edge $(p_i, p_j) \in E_{(p,p)}$) {
2.     $P_i = compose\_predicate(p_i)$; /* recursive call */
3.     $P_j = compose\_predicate(p_j)$; /* recursive call */
4.     if ($f_{Epp}((p_i, p_j))$ == '$\wedge$') { $P = (P_i \wedge P_j)$; }
5.     else { $P = (P_i \vee P_j)$; }
6.     return($P$);
7. } else { /* there is only one oval node $p$ in $LF$ */
8.     $P = f_{Np}(p)$;
9.     return($P$);
10. }
```

More formally, the transformation process can be stated by the following function — *LF_to_RA_without_DN*. Notice that before calling *LF_to_RA_without_DN*, all pseudo predicates are assumed to be deleted and their corresponding attributes collected in $T(Q)$.

FUNCTION **4.2** *LF_to_RA_without_DN(LF)*

Input: a logical form, $LF = (N_e \cup N_p, E, f_N, f_E)$, containing no diamond nodes.
Output: the corresponding Relational Algebra Expression RAE.

```
1. if (there is an edge $(e_i, e_j) \in E_{(e,e)}$) {
2.     decompose $LF$ into $LF_i$ and $LF_j$;
3.     $RAE_i = LF\_to\_RA\_without\_DN(LF_i)$; /* recursive call */
4.     $RAE_j = LF\_to\_RA\_without\_DN(LF_j)$; /* recursive call */
5.     if ($f_{Eee}((e_i, e_j))$ == '$\wedge$') { $RAE = (RAE_i \cap RAE_j)$; }
6.     else { $RAE = (RAE_i \cup RAE_j)$; }
7.     return($RAE$);
8. } else { /* there is only one rectangle node $e$ in $LF$ */
9.     $P = compose\_predicate((N_p, E_{(p,p)}, f_{Np}, f_{Epp}))$;
10.    $RAE = \pi_{T(Q)}(\sigma_P(\pi_1(f_{Ne}(e))))$;
11.    return($RAE$);
12. }
```

4.2 The Transformation Process for a Logical Form Containing One or More Diamond Nodes

In this section, we distinguish the transformation processes for a logical form containing one or more diamond nodes.

4.2.1 The Transformation Process for a Logical Form Containing One Diamond Node

Without loss of generality, we first assume that the logical form has no edge of $E_{(e,e)}$. If there are edges of $E_{(e,e)}$ then the logical form can be decomposed according to these edges and the answer is the union/intersection of the results of the sub-logical forms. For example, the logical form in Figure 11(b) can be decomposed into the following two sub-logical forms.

A relational algebra expression represents an execution order of the query. The transformation of a query Q to a relational algebra expression is essentially to determine an algebraic order from its logical form.

The relationship set relates a set of entity sets. That is, the surrogate values of the entity relations can be used to semi-join (i.e. to restrict [1]) the surrogate values of the relationship relation to produce the *answer surrogate values* (which will be used to compute the answer). Before performing the semi-joins, different conditions such as the one-to-all relationship (Section 3.1.3), the negative form relationship (Section 3.1.4), and the predicates (Section 3.1.1) can be evaluated to reduce the surrogate values of the relationship set and entity sets.

The predicates can be directly applied to restrict the corresponding relationship relation or entity relations first. A one-to-all relationship can be implemented by a division operation [10]. Moreover, a negative form relationship can be implemented by computing the affirmative form relationship followed by a set difference operation.

The transformation process can therefore be stated in the following five phases.

1. For each $e_i \in N_e$ and the $r \in N_r$, use their respective predicates, except for pseudo predicates, to restrict the corresponding entity and relationship relations. That is, the transformation first produces $\sigma_{P_i}(\pi_1(f_{Ne}(e_i)))$ and $\sigma_P(\pi_1(f_{Nr}(r))), \forall e_i \in N_e$ and $r \in N_r$, where P_i and P are the compound predicates of the relations $\pi_1(f_{Ne}(e_i))$ and $\pi_1(f_{Nr}(r))$, respectively. P_i and P can be obtained by *compose_predicate*.

2. Project the surrogates of all relations. Let S_i denotes the set of surrogate values of the restricted entity relation RE_i obtained in Phase (1), and $(S_1, S_2, \ldots, S_n)$ denotes the set of surrogate values of the restricted relationship relation $RR_{(1,2,\ldots,n)}$ which associates with the entity relations RE_1, $RE_2, \ldots,$ and RE_n.

3. If there are some entity relations which correspond to shadow rectangle nodes (i.e., there exist one-to-all relationships), say S_1, $S_2, \ldots, S_k$, then define

$$< S_{k+1}, S_{k+2}, \ldots, S_n > \equiv (\cdots (((S_1, S_2, \ldots, S_n) \div S_1) \div S_2) \cdots) \div S_k.$$

   Also, define

$$[S_1, S_2, \ldots, S_n] \equiv (S_1, S_2, \ldots, S_n) \overset{S_{k+1}, S_{k+2}, \ldots, S_n}{\ltimes} < S_{k+1}, S_{k+2}, \ldots, S_n > .$$

   Otherwise, define

$$[S_1, S_2, \ldots, S_n] \equiv (S_1, S_2, \ldots, S_n).$$

4. If the diamond node corresponding to the relationship relation has a black triangle linked to an entity relation (i.e., there exist negative form relationships), say S_i, then the set of the answer surrogate values can be defined as

$$\subset S_1, S_2, \ldots, S_n \supset \equiv (S_1, S_2, \ldots, S_n) \overset{S_i}{\ltimes} (S_i - \pi_{S_i}([S_1, S_2, \ldots, S_n])),$$

   else

$$\subset S_1, S_2, \ldots, S_n \supset \equiv [S_1, S_2, \ldots, S_n].$$

5. Since $\subset S_1, S_2, \ldots, S_n \supset$ contains the answer surrogate values, we can join these values with the surrogate values of all relations and project the target attributes to get the answer. That is, the answer of the query can be produced by

$$\pi_{T(Q)}(\subset S_1, S_2, \ldots, S_n \supset \overset{S_1,S_2,\ldots,S_n}{\bowtie} RR_{(1,2,\ldots,n)} \overset{S_1}{\bowtie} RE_1 \overset{S_2}{\bowtie} RE_2 \cdots \overset{S_n}{\bowtie} RE_n).$$

However, we can often eliminate unnecessary join operations. If, for example, the attributes of RE_i is not contained in $T(Q)$ then the join on RE_i can be eliminated.

We illustrate these five phases by the following function.

FUNCTION **4.3** *LF_to_RA_without_Eee(LF)*

Input: a logical form with $E_{(e,e)} = \phi$.
Output: the corresponding Relational Algebra Expression RAE.

```
 1. for each $e_i \in N_e$ { /* assume i = 1, 2,..., n */
 2.     $RE_i = \sigma_{P_i}(\pi_1(f_{Ne}(e_i)))$; /* $P_i$ can be obtained by compose_predicate */
 3. }
 4. $RR_{(1,2,\ldots,n)} = \sigma_P(\pi_1(f_{Nr}(r)))$; /* $P$ can be obtained by compose_predicate */
 5. $S_i = \pi_{surrogate}(RE_i), \forall i = 1, 2, \ldots, n$;
 6. $(S_1, S_2, \ldots, S_n) = \pi_{surrogate}(RR_{(1,2,\ldots,n)})$;
 7. if (there exists $e_1, e_2, \ldots, e_k$, $k \neq 0$, such that $\pi_2(F_{Ne}(e_i)) =$ '$\forall$', $1 \leq i \leq k$) {
 8.     $< S_{k+1}, S_{k+2}, \ldots, S_n >= (\cdots(((S_1, S_2, \ldots, S_n) \div S_1) \div S_2)\cdots) \div S_k$;
 9.     $[S_1, S_2, \ldots, S_n] = (S_1, S_2, \ldots, S_n) \overset{S_{k+1},S_{k+2},\ldots,S_n}{\ltimes} < S_{k+1}, S_{k+2}, \ldots, S_n >$;
10. } else { $[S_1, S_2, \ldots, S_n] = (S_1, S_2, \ldots, S_n)$; }
11. if (there is an edge $(r, e_i)$ such that $\pi_2(f_{Ere}((r, e_i))) =$ '$N$') {
12.     $\subset S_1, S_2, \ldots, S_n \supset = (S_1, S_2, \ldots, S_n) \overset{S_i}{\ltimes} (S_i - \pi_{S_i}([S_1, S_2, \ldots, S_n]))$;
13. } else { $\subset S_1, S_2, \ldots, S_n \supset = [S_1, S_2, \ldots, S_n]$; }
14. $RAE = \pi_{T(Q)}(\subset S_1, S_2, \ldots, S_n \supset \overset{S_1,S_2,\ldots,S_n}{\bowtie} RR_{(1,2,\ldots,n)} \overset{S_1}{\bowtie} RE_1 \overset{S_2}{\bowtie} RE_2 \cdots \overset{S_n}{\bowtie} RE_n)$;
15. return($RAE$);
```

Now, if there are edges of $E(e, e)$ then, by decomposing the logical form into sub-logical forms, the function *LF_to_RA_without_Eee* can be employed to process these sub-logical forms separately. The whole process can be specified by the following function.

FUNCTION **4.4** *LF_to_RA_with_one_DN(LF)*

Input: a logical form containing one diamond node.
Output: the corresponding Relational Algebra Expression RAE.

```
1.  if (there is an edge $(e_i, e_j) \in E_{(e,e)}$) {
2.      decompose $LF$ into $LF_i$ and $LF_j$;
3.      $RAE_i$ = $LF\_to\_RA\_with\_one\_DN(LF_i)$; /* recursive call */
4.      $RAE_j$ = $LF\_to\_RA\_with\_one\_DN(LF_j)$; /* recursive call */
5.      if ($f_{Eee}((e_i, e_j))$ == '$\wedge$') { $RAE = (RAE_i \cap RAE_j)$; }
6.      else { $RAE = (RAE_i \cup RAE_j)$; }
7.      return($RAE$);
8.  } else { /* there is no $(e_i, e_j)$ */
9.      $RAE$ = $LF\_to\_RA\_without\_Eee(LF)$;
11.     return($RAE$);
12. }
```

4.2.2 The Transformation Process for a Logical Form Containing More Than One Diamond Node

If a logical form contains more than one diamond node linked by edges of $E_{(r,r)}$ then, without loss of generality, it can be decomposed into sub-logical forms according to the edges. The sub-logical forms can be separately transformed by the process presented in Section 4.2.1. The final answer is the union/intersection of the results of these sub-logical forms. For example, the logical form in Figure 11(c) can be decomposed into the following two sub-logical forms.

The result of this decomposition is equivalent to that of Figure 11(b). This is because the query corresponding to Figure 11(b) has the same semantic meaning as the query corresponding to Figure 11(c). Therefore, in such cases, the transformation process is similar to *LF_to_RA_with_one_DN*, except that "$(e_i, e_j) \in E_{(e,e)}$" (Step 1) and "$f_{Eee}((e_i, e_j))$" (Step 5) need to be replaced by "$(r_i, r_j) \in E_{(r,r)}$" and "$f_{Err}((r_i, r_j))$", respectively.

If a logical form containing more than one diamond node but these nodes are not linked by edges of $E_{(r,r)}$, say $\boxed{E_1} < R_1 > \boxed{E_2} < R_2 > \boxed{E_3} \cdots \boxed{E_n} < R_n > \boxed{E_{n+1}}$, where $\boxed{E_i}$ and $< R_j >$ represent rectangle nodes and diamond nodes, respectively. Then, assume the pseudo predicate is on $\boxed{E_{n+1}}$, it can be organized into a nested relationship,

$$NR \equiv \boxed{\cdots \boxed{\boxed{\boxed{E_1} < R_1 > \boxed{E_2}} < R_2 > \boxed{E_3}} \cdots \boxed{E_n}} < R_n > \boxed{E_{n+1}},$$

where $\boxed{E_2}$, $\boxed{E_3}$,..., $\boxed{E_n}$, are all attached with the pseudo predicate "surrogate = ?". Then NR can be transformed as follows.

FUNCTION **4.5** *NR_to_RA(LF)*

Input: a nested relationship $NR = \boxed{E'_n} < R_n > \boxed{E_{n+1}}$.

Output: the corresponding Relational Algebra Expression RAE.

```
1. if ($\boxed{E'_n}$ is nested) {
2.     $RAE$ = $NR\_to\_RA(E'_n)$; /* recursive call */
3.     $RAE$ = $LF\_to\_RA\_with\_one\_DN(\boxed{E'_n} < R_n > \boxed{E_{n+1}})$;
4. } else { $RAE$ = $LF\_to\_RA\_with\_one\_DN(\boxed{E'_n} < R_n > \boxed{E_{n+1}})$; }
5. return($RAE$);
```

For example, in Section 2.2.2, the logical form $\boxed{\text{Teacher}}$<Teach>$\boxed{\text{Course}}$<Take>$\boxed{\text{Student}}$ can be organized into

$$\boxed{\boxed{\text{Teacher}} < \text{Teach} > \boxed{\text{Course}}} < \text{Take} > \boxed{\text{Student}}.$$

Thus, *NR_to_RA* first transforms $\boxed{\text{Teacher}}$<Teach>$\boxed{\text{Course}}$ then treats the answer as an entity $\boxed{E}$ by projecting surrogate of $\boxed{\text{Course}}$, and finally transform $\boxed{E}$<Take>$\boxed{\text{Student}}$ into the corresponding relational algebra expression.

In the following, we follow Example 3.1 (Figure 16) to show the kernel transformation process — *LF_to_RA_without_Eee*. Refer to Figure 3 for this example.

EXAMPLE **4.1** In Figure 16, we depict the logical form $LF(Q) = (N, E, f_N, f_E)$, where Q = "List the London suppliers who *do not* supply *all* red parts.", and we have presented $LF(Q)$ in Example 3.1 in detail. The transformation process is now explained as follows. Notice that we also evaluate the algebraic operations in the process.

1. After restricting both entity relations Suppliers and Parts, we obtain

$$RE_1 \equiv \sigma_{city=London}(\text{Suppliers}) \text{ and } RE_2 \equiv \sigma_{color=red}(\text{Parts}).$$

   Note that there is no restriction on Shipments. Therefore, $RR_{(1,2)} \equiv$ Shipments.

2. Perform the projections on surrogates, we get the surrogates as follows.

$$\begin{aligned} &S_1 \equiv \pi_{sno}(RE_1) = \pi_{sno}(\sigma_{city=London}(\text{Suppliers})) = \{S1, S4\},\\ &S_2 \equiv \pi_{pno}(RE_2) = \pi_{pno}(\sigma_{color=red}(\text{Parts})) = \{P1, P4, P6\}, \text{ and}\\ &(S_1, S_2) \equiv \pi_{sno,pno}(RR_{(1,2)}) = \pi_{sno,pno}(\text{Shipments}) = \{(S1, P1), (S1, P2),\\ &\quad (S1, P3), (S1, P4), (S1, P5), (S1, P6), (S2, P1), (S2, P2), (S3, P2),\\ &\quad (S4, P2), (S4, P4), (S4, P5)\}. \end{aligned}$$

3. $< S_1 > \equiv (S_1, S_2) \div S_2$ and

$$\begin{aligned} [S_1, S_2] &\equiv (S_1, S_2) \overset{S_1}{\ltimes} < S_1 > = (S_1, S_2) \overset{S_1}{\ltimes} ((S_1, S_2) \div S_2)\\ &= (S_1, S_2) \overset{S_1}{\ltimes} (\{S1\})\\ &= \{(S1, P1), (S1, P2), (S1, P3), (S1, P4), (S1, P5), (S1, P6)\}. \end{aligned}$$

4. $$\begin{aligned} \subset S_1, S_2 \supset &\equiv (S_1, S_2) \overset{S_1}{\ltimes} (S_1 - \pi_{S_1}([S_1, S_2]))\\ &= (S_1, S_2) \overset{S_1}{\ltimes} (\{S1, S4\} - \{S1\})\\ &= (S_1, S_2) \overset{S_1}{\ltimes} (\{S4\})\\ &= \{(S4, P2), (S4, P4), (S4, P5)\}. \end{aligned}$$

5. Because there is only one attribute Suppliers.sname in $T(Q)$, we eliminate unnecessary join operations and perform

$$\begin{aligned} &\pi_{sname}(\subset S_1, S_2 \supset \overset{sno}{\bowtie} RE_1)\\ &= \pi_{sname}((\{(S4, P2), (S4, P4), (S4, P5)\}) \overset{sno}{\bowtie} \sigma_{city=London}(\text{Suppliers}))\\ &= \{ \text{ Clark } \}. \qquad \square \end{aligned}$$

5 Conclusions and Future Research

We study the inter-relationship between natural language constructs and the E-R conceptual schema. The basic parts of English sentences can be mapped into E-R schemas in a natural way. If the underlying schema of a database was pre-existing but not structured by E-R approach, then the schema can be restructured by E-R approach through defining Entity-Relationship views to achieve our work.

We develop a logical form by extending the E-R representations to capture natural language semantics and describe a processing model for the query transformation process. In this processing model, when the target database is changed we only have to change the dictionary and the ER/SRF. The front-end parser/mapper and the query transformer remain unchanged.

English sentences may also be mapped into the universal relation [19, 20], in which the entire database is imagined to be kept in a single relation. But this needs to further transform the universal query command into the actual stored schema. Moreover, other intermediate forms suffer from the bias to natural language constructs and much effort is needed to transform them into database query languages. In comparison, our logical form has the merits that not only can it be mapped from natural language constructs but also it is represented in a form similar to the ERD and can be efficiently transformed into the relational algebra.

Finally, an extension for mapping natural language constructs into the schema generated by the Extended E-R (EER) [17] or an object-oriented design methodology (e.g. the one proposed by Blaha, et al. [2]) will be investigated in the near future. Besides, by combining the framework presented by Zvieli and Chen [25], our work can be extended to process a natural language query involving a modifier like 'almost', 'very', or 'nearly'. This combination is served as a step toward analyzing the use of modifiers, which are fuzzy in natural, to communicate with fuzzy databases.

Acknowledgement

The authors wish to thank the anonymous referees whose invaluable comments and suggestions helped to improve this paper substantially.

References

[1] P.A. Bernstein and D.M.W. Chiu, Using Semi-Joins to Solve Relational Queries, *Journal of the Association for Computing Machinery* 28 (1) (1981) 25-40.

[2] M.R. Blaha, W.J. Premerlani and J.E. Rumbaugh, Relational Database Design Using an Objected-Oriented Methodology, *Comm. ACM* 31 (4) (1988) 414-427.

[3] P.P. Chen, The Entity-Relationship Model — Toward a Unified View of Data, *ACM Trans. Database Systems* 1 (1) (1976) 9-36.

[4] P.P. Chen, English Sentence Structure and E-R Diagrams, *Information Sciences* 29 (2) (1983) 127-149.

[5] A.L.P. Chen, et al., Distributed Query Processing in a Multiple Database System, *IEEE Journal on Selected Areas in Communications* 7 (3) (1989) 390-398.

[6] A.L.P. Chen, A Localized Approach to Distributed Query Processing, *Lecture Notes in Computer Science: Advances in Database Technology — EDBT'90* (416, Springer-Verlag, Berlin, 1990) 188-202.

[7] F.J. Damerau, Problems and Some Solutions in Customization of Natural Language Database Front Ends, *ACM Trans. Office Information Systems* 3 (2) (1985) 165-184.

[8] C.J. Date, *A Guide to INGRES* (Addison-Wesley, MA, 1987).

[9] C.J. Date, *A Guide to the SQL Standard* (Addison-Wesley, MA, 1989).

[10] C.J. Date, *An Introduction to Database Systems* (Addsion-Wesley, MA, 5th ed., 1990).

[11] B.J. Grosz, et al., TEAM: An Experiment in the Design of Transportable Natural-Language Interfaces, *Artificial Intelligence* 32 (1) (1987) 173-243.

[12] G. Jakobson et al., An Intelligent Database Assistant, *IEEE Expert* 1 (2) (1986) 65-79.

[13] R.A. Kowalski, *Logic for Problem Solving* (North-Holland, Amsterdam, 1979).

[14] M.C. McCord, Using Slots and Modifiers in Logic Grammars for Natural Language, *Artificial Intelligence* 18 (3) (1982) 327-367.

[15] F.C.N. Pereira and D.H.D. Warren, Definite Clause Grammars for Language Analysis — A Survey of the Formalism and a Comparison with Augmented Transition Networks, *Artificial Intelligence* 13 (3) (1980) 231-278.

[16] G. Piatetsky-Shapiro and G. Jakobson, An Intermediate Database Language and Its Rule-based Transformation to Different Database Languages, *Data & Knowledge Engineering* 2 (1987) 1-29.

[17] T.J. Teorey, D.Yang, and J.P. Fry, A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model, *ACM Computing Surveys* 18 (2) (1986) 197-222.

[18] F.S.C. Tseng, S.Y. Lee and W.P. Yang, DELICIOUS: An Intermediate Code Scheme for Heterogeneous Database Systems, *Proc. International Computer Symposium,* Taiwan, ROC (1988) 998-1003.

[19] J.D. Ullman, *Principles of Database Systems* (Computer Science Press, Rockville, MD, 2nd ed., 1982).

[20] J.D. Ullman, *Principles of Database and Knowledge-Base Systems* (Computer Science Press, Vol. 2, Rockville, MD, 1988).

[21] P. Velardi, Natural Language Interfaces to Databases: Features and Limitations, *Proc. 7th Int'l Conf. Entity-Relationship Approach — A Bridge to the User* (1988).

[22] M. Wallace, *Communicating with Databases in Natural Language* (Ellis Horwood, England, 1984).

[23] P.H. Winston, *Artificial Intelligence* (Addison-Wesley, MA, 1984).

[24] W.A. Wood, Transition Network Grammars for Natural Language Analysis, *Comm. ACM* 13 (10) (1970) 591-606.

[25] A. Zvieli and P.P. Chen, Entity-Relationship Modeling and Fuzzy Databases, *Proc. IEEE Int. Conf. Data Engineering* (1986) 320-327.