SUPPORTING NATURAL LANGUAGE INTERFACES FOR RELATIONAL DATABASES USING RRA-A RESHAPED RELATIONAL ALGEBRA

by

Yoav Raz

Technical Report #410
May 1986
SUPPORTING NATURAL LANGUAGE INTERFACES FOR RELATIONAL DATABASES USING RRA - A RESHAPED RELATIONAL ALGEBRA

by

Yoav Raz
Computer Science Department
Technion - Israel Institute of Technology
Haifa 32000, Israel

ABSTRACT

A methodology based on RRA for constructing portable Natural Language (NL) interfaces for relational databases is presented. This methodology differs from other approaches in assuming that for database NL queries, a "rough" analysis is sufficient in most cases to extract the intended meaning, and hence, bypassing many difficult problems of NL processing.

RRA - the Reshaped Relational Algebra is equivalent to the regular Relational Algebra (RA), but has different operators. Its operators, a modification of the RA's operators, are intended to follow the semantics of queries phrased in Natural English Language (NL). For each operator there are appropriate elementary syntactical constructs of natural language. These elementary constructs can be combined to form complex sentences (queries), whose semantics is given by algebraic expressions composed of corresponding operators. The above set of syntactical constructs provides the elements for the "rough" analysis of NL queries and proves itself as rich enough to express most queries that "make sense". As such, the RRA is useful both in defining semantics for Natural Language subsets or languages alike, and in implementing efficiently NL translators for interfaces to relational databases.

Another important idea behind the presented methodology is the usage of a restricted terminology for a given database. The "knowledge" behind the database is expressed by a set of surface "simple" sentences (describing relationships among entities and attribute values) that give the terminology. Being aware of the above restrictions, the user can phrase complex queries flexibly. Following the surface sentences the system can relatively easily map a query into an RRA expression.

The most important feature of the methodology is its simplicity. It can be implemented on contemporary personal computers, and can be adopted to any "well defined" relational database by specifying its surface sentences, and their mapping to its relations.

The RRA is presented and its connection to Natural Language is explained. It is also demonstrated how to map complex NL sentences into RRA expressions. The presentation here does not depend on a specific NL subset or NL-like language, and the idea can be implemented via various kinds of language processors.
1. INTRODUCTION

A methodology based on RRA (a Reshaped Relational Algebra - [MR2], [MR3]) for constructing Natural Language (NL) interfaces for relational databases is presented. This methodology differs from other approaches in assuming that for database NL queries, a "rough" analysis is sufficient in most cases to get the correct meaning, and hence, bypassing many difficult problems of NL processing. The other approaches are based on a detailed analysis of NL constructs (e.g. [BLT], [BMS], [DAM], [EPS], [GRO], [HAR], [HSS], [LEH], [TT], [WAL]).

The RRA was originally defined as a tool for a compact definition of the semantics of ERROL (an Entity Relationship Role Oriented Language - see [MR1], [RCM]) which is an English-like query language over the Entity Relationship Model (ERM, see [CHE]). In this paper RRA is redefined (following [RAZ]) in order to extend its range of application.

The analogies to Natural Language carried by the RRA are exposed when we use distinct relations to describe both entity (object) sets and relationship (association) sets. An advantage is taken of the fact that relationships are usually described by a "simple" sentence (see [BIL]). A "closed world" ([REI]) is also assumed.

RRA is equivalent ([RAZ]) to the regular Relational Algebra (RA), but has different operators. Its operators, a modification of the RA's operators, are intended to follow the semantics of queries phrased in Natural English Language (NL). For each operator there are appropriate elementary syntactical constructs of natural language. These elementary constructs can be combined to form complex sentences (queries), having semantics given by algebraic expressions composed of corresponding operators. The above set of syntactical constructs provides the elements for the "rough" analysis of NL queries and proves itself as rich enough to express most queries that "make sense". As such, the RRA is useful both in defining semantics for Natural Language subsets or languages alike, and in implementing efficiently NL translators for interfaces to relational databases. What makes RRA more convenient for this purpose than RA is the strict correspondence between syntactical elements and operators. Using standard RA the mapping should be done to (sometimes complicated) RA expressions (rather than operators), but of course, it can be done. A second reason is implementation related; since what is needed is exactly the RRA operations, it is worthwhile concentrating on an efficient implementation of RRA operators and their combinations.
Another important idea behind the presented methodology is the usage of a restricted terminology for a given database. The scope of the terminology is determined as follows:
- The accumulated knowledge in the database is expressed by a set of "simple" sentences, the "surface sentences", describing relationships among entities and attributes.
- Entity type names in these simple sentences are distinct.
- NL (or NL like) queries are expressed using only entity type names, attribute names and predicates taken from the above set with the addition of several other terms like quantifiers, logical connectives, comparison expressions etc.

Being aware of the above restrictions, the user can phrase complex queries flexibly. Following the surface sentences of the accumulated knowledge the system can relatively easily map a query into an RRA expression. Almost all the information needed by the system concerns the correct navigation in the database, and this can be done by matching portions of the query to surface sentences in certain legitimate orders. In many cases the match can be loose (up to the roots of words). A mismatch indicates an error in the query. Hence, according to our approach it is enough for the system to validate a certain "correct" skeleton of the query, and not to deal with all nuances of NL.

The most important message of the methodology is its simplicity. It can be implemented on contemporary personal computers, and can be adopted to any ER-consistent (MMR) relational database by specifying its surface sentences, and their mapping to its relations. Hence, the methodology provides a basis for constructing domain portable systems.

In this paper the RRA is presented and its connection to Natural Language is explored. It is also demonstrated how to map complex NL sentences into RRA expressions. The presentation here does not depend on a specific NL subset or NL-like language, and the idea can be implemented via various types of language processors. Part 2 gives a formal definition of the RRA (following [RAZ]) and Part 3 discusses the linguistic aspect of the DB information. In Part 4 the analogy between RRA and Natural Language constructs is explained and demonstrated. Part 5 describes how to construct RRA expressions for complex NL sentences. Part 6 deals with references in NL sentences. In Part 7 query and database manipulations are discussed. Part 8 gives some conclusions and an application of the methodology is demonstrated in the Appendix via ERROL ([RCM],[IR]) examples.
2. THE RRA - RESHAPED RELATIONAL ALGEBRA

The RRA is equivalent to the RA and has similar operators. Some of its operators are identical to those of RA and some of them are extensions of the RA's (See [PIR]). The main difference lies in an implicit join operation embedded in binary operations. This embedded join is due to references ("the above", "that", "the mentioned") which appear in NL (see below, Part 6). A different operator is the NOT (complement) operator which expresses NL's negation. This operator enables RRA to express the RA's subtraction.

Operators which extend the scope of the usual RA are the AGGRAGATE FUNCTION operators which can be viewed as both RRA's and RA's.

2.1 NOTATIONS AND DEFINITIONS

A relation consists of a name, structure and value.

Notation: \( S(R), V(R) \) are the structure and value, respectively, of a relation with the name \( R \).

Let \( A_1, A_2, \ldots, A_n \) be the attribute names of \( R \) and \( D_{A_1}, D_{A_2}, \ldots, D_{A_n} \) their corresponding domains (a domain is any set of objects of the same type).

Let also

\[
A = \{A_1, A_2, \ldots, A_n\}.
\]

Then, \( R \) is denoted also as \( R(A) \);

\[
S(R) = \{x : D_x | x \in A\} = \{A_i : D_{A_i} | i = 1, \ldots, n\};
\]

\[
V(R) = \{t : A \rightarrow \bigcup_{x \in A} D_x | t \text{ (a "tuple") is a total function over } A \text{, which maps each } x \in A \text{ to an element in } D_x\};
\]

In the following sections the RRA operations are defined. The result relation of any operation will be denoted \( R', S', V' \) for name, structure and value respectively.
2.2 RENAMING

Let $R = R(A)$

$B = \{B_1, B_2, ..., B_s\}$ a set of attribute names.

$M : A \rightarrow B$ such that $M(A_i) = B_j, i = 1, 2, ..., n$.

Then

$R' = \text{rename}_M(R)$;

$S' = S(R') = \{M(x): D_x | x \in A\} = \{B_i: D_{A_i} | i = 1, ..., n\}$

$V' = V(R') = \{t': M(A) \rightarrow \bigcup_{x \in A} D_x | (\exists t \in V(R))(\forall x \in A)(t'(M(x)) = t(x))\}$.

2.3 PROJECTION

Let $R = R(A)$

$A' \subset A$.

Then

$R' = R [A']$ or $R' = \text{project}_{A'}(R)$

$S' = S(R') = \{x: D_x | x \in A'\}$

$V' = V(R') = \{t': A' \rightarrow \bigcup_{x \in A'} D_x | (\exists t \in V(R))(\forall x \in A')(t'(x) = t(x))\}$. 
2.4 SELECTION

Let \( R = R(A \cup \{a\}) \)
\[ a \notin A \]
\[ c = \text{const.} \]

Then
\[ R' = \text{select}_{a \leq c}(R) \]
where \( \theta \in \{=, \neq, <, >, \leq, \geq\} \)
\[ S' = S(R') = \{x : D_x | x \in A\} \]
\[ V' = V(R') = \{t' : A \rightarrow \bigcup D_x | (\exists t \in V(R))((t' = t[A]) \land (a) = c)\} \]
where \( t[A] \) is the restriction of \( t \) on the domain \( A \).

2.5 PRODUCT (NATURAL JOIN, CARTESIAN PRODUCT, INTERSECTION)

Let \( R_1 = R_1(A \cup B), \ R_2 = R_2(B \cup C) \)
where \( A, B, C \) disjoint.

Then
\[ R' = R_1 \times R_2 \]
\[ S' = S(R') = S(R_1) \cup S(R_2) = \{x : D_x | x \in A \cup B \cup C\} \]
\[ V' = V(R') = \{t' : A \cup B \cup C \rightarrow \bigcup D_x | (\exists t_1 \in V(R_1)) \land (\exists t_2 \in V(R_2))(t'[A \cup B] = t_1 \land t'[B \cup C] = t_2)\} \]
where \( t'[A'] \) is the restriction of \( t' \) on domain \( A' \subset A \cup B \cup C \).

REMARKS:
1) This operation is equivalent to the regular Relational Algebra Natural join over \( B \).
2) If \( B = \emptyset \) it reduces to the regular Cartesian Product.
3) If \( A = C = \emptyset \) it reduces to the regular Intersection.
2.6 BORDERED UNION or OR

Let \( R_1 = R_1(A \cup B) \), \( R_2 = R_2(B \cup C) \)

where \( A, B, C \) disjoint.

Then

\[
R' = R_1 \text{ or } R_2 \\
S' = S(R_1) \cup S(R_2) = \{x : D_x | x \in A \cup B \cup C\} \\
V' = V(R_1) = \{t' : A \cup B \cup C \rightarrow \bigcup_{x \in A \cup B \cup C} D_x | \exists t_1 \in V(R_1) \exists t_2 \in V(R_2) \}
\]

\[
(t'[A \cup B] = t_1[A \cup B] \land t'[B \cup C] = t_2[B \cup C] \land t_1(a_1) \theta t_2(a_2))
\]

REMARK: If \( A = C = \emptyset \) it reduces to the regular Union.

2.7 ATTRIBUTE JOIN or \( \theta \)-JOIN

Let \( R_1 = R_1(A \cup B \cup \{a_1\}) \), \( R_2 = R_2(B \cup C \cup \{a_2\}) \)

where \( A, B, C \) disjoint.

\( a_1, a_2 \not\in A \cup B \cup C \)

\( D_{a_1}, D_{a_2} \) have elements of the same type.

Then

\[
R' = R_1 \text{ join } R_2 \\
\theta \in \{=, \lt, \leq, \gt, \geq\} \\
S' = S(R_1) \cup S(R_2) = \{x : D_x | x \in A \cup B \cup C\} \\
V' = V(R_1) = \{t' : A \cup B \cup C \rightarrow \bigcup_{x \in A \cup B \cup C} D_x | \exists t_1 \in V(R_1) \exists t_2 \in V(R_2) \}
\]

\[
(t'[A \cup B] = t_1[A \cup B] \land t'[B \cup C] = t_2[B \cup C] \land \theta(a_1) \theta(a_2))
\]

REMARK: If \( B = \emptyset \) it reduces to the regular \( \theta \)-Join.
2.8 SET-JOIN or GENERALIZED DIVISION

Let \( R_1 = R_1(A \cup B \cup D_1) \), \( R_2 = R_2(B \cup C \cup D_2) \)

where \( A, B, C, D_1, D_2 \) are disjoint and

there exists a one-one-on mapping \( f \)

\( f : D_1 \rightarrow D_2 \)

such that \( f(x_1) = x_2, x_1 \in D_1, x_2 \in D_2 \) if and only if

\( D_{x_1}, D_{x_2} \) have elements of the same type.

Then

\[ R' = R_1 \text{ set-join } R_2 \text{ where } \delta \in \{ =, \neq, >, <, \geq, \leq \} \]

\[ S' = S(R') = \{ x : D_2 | x \in A \cup B \cup C \} \]

\[ V' = V(R') = \{ t' : A \cup B \cup C \rightarrow \bigcup_{x \in A \cup B \cup C} D_2 | \]

\[ (\exists t_1 \in V(R_1) \exists t_2 \in V(R_2)) (t'[A \cup B] = t_1[A \cup B] \land t'[B \cup C] = t_2[B \cup C] \land \]

\[ \land V((\text{select}_{A \cup B = t_1(A \cup B)}(R_1))[D_1]) \delta V((\text{select}_{B \cup C = t_2(B \cup C)}(R_2))[D_2])) \]

where \( \text{select}_{A'} \) of \( R(A) \) over every \( x \in A' \).

\( \text{select}_{x \in (t)} \) of \( R(A) \) over every \( x \in A' \).

\textbf{REMARK:} If \( B = \emptyset \) it reduces to a regular Generalized Division.
2.9 COMPLEMENT or NOT

Let \( R = R(A), R_x = R_x((x)) \) and \( x \in A' \subset A \).

Then
\[
R' = \text{not}(R \mid x \in A') \ (R)
\]
\[
S' = S(R') = \{x : D_x \mid x \in A'\}
\]
\[
V' = V(R') = \{t' : A' \rightarrow \bigcup D_x \mid (\exists \ t_x \in V(R)) (t'(x) = t_x(x)) \wedge t' \in V(R(A'))\} =
\]
\[
\times V(R_x) - V(R(A'))
\]

where "\( \times \)" stands for the regular Cartesian Product, and "-" stands for the regular Subtraction.

2.10 AGGREGATE FUNCTIONS (AFs)

Let \( R = R(A \bigcup \{a\}), a \notin A \).

\[
AF_a : \mathcal{P}\_a \rightarrow D_{AF\_a}
\]

where \( AF \in \{\text{SUM, COUNT, MAX, MIN} \ldots\} \).

Then
\[
R' = \text{AF}_a(R)
\]
\[
S' = S(R') = \{x : D_x \mid x \in A' = A \bigcup \{AF\_a\}\}
\]
\[
V' = V(R') = \{t' : A' \rightarrow \bigcup D_x \mid (\exists \ t_x \in V(R)) (t'(x) = t_x(x)) \wedge t' \in V(R(A'))\} =
\]
\[
\times V(R_x) - V(R(A'))
\]

REMARK: When \( AF = \text{SUM} \), the "select" and "project" should be modified; its results are multisets rather than sets.
3. THE LINGUISTIC ASPECT OF THE INFORMATION IN THE DATABASE

It is common among DB designers to describe databases on its conceptual level using Entity (Object) - Relationship (association) Diagrams (ERDs, [CHE]) which are a kind of semantic networks ([BF]) used for knowledge representation. Informally, an ERD is a graph with three types of nodes:

1) Entity sets. 2) Relationship sets. 3) Attributes.

The above sets consist of elements of the same type (with the same attributes). The arcs connect entity sets with appropriate attributes and relationship sets, and also relationship sets with their attributes. An example of an ERD appears in Fig. 1. The notion of ERD can be extended beyond what demonstrated here to capture more of the real world's semantics. However, even with its simple version the basic ideas can be presented.

A relationship between entities can usually be described by a simple sentence with an entity as the subject part, the other entities participating in the relationship as objects and a predicate part which usually induces the relationship's name.

For example, a member in the relationship set SUPPLY can be described by the sentence

SUPPLIER SUPPLIES ITEM to DEPARTMENT. or
DEPARTMENT IS SUPPLIED by SUPPLIER with ITEM.

If the relationship has attributes, they may be combined in the sentence:

SUPPLIER STOCKS QUANTITY of ITEM.

The association between an entity and its attribute may be expressed as

SUPPLIER HAS NAME... or SUPPLIER IS NAMED...
NAME OF SUPPLIER... or SUPPLIER's NAME...

The RRA has been designed to take advantage of the ERD's linguistic aspect by calculating relations which contain tuples satisfying predicates expressed in natural language sentences based on the sentences derived from the ERD (the surface sentences).
Fig. 1

The rectangular boxes represent entity sets; diamonds - relationship sets; round boxes - attributes.

\[ \text{DEPARTMENT} = \text{DEPARTMENT (No, NAME, FLOOR)}; \text{ key=No}. \]
\[ \text{ITEM} = \text{ITEM (No, NAME, COLOR, TYPE)}; \text{ key=No}. \]
\[ \text{SUPPLIER} = \text{SUPPLIER (No, NAME, LOCALITY)}; \text{ key=No}. \]
\[ \text{REQUEST} = \text{REQUEST (DEPARTMENT -key, ITEM -key, QUANTITY)} \]
\[ \text{STOCK} = \text{STOCK (ITEM -key, SUPPLIER -key, QUANTITY)} \]
\[ \text{SUPPLY} = \text{SUPPLY (ITEM -key, SUPPLIER -key, DEPARTMENT -key, QUANTITY, PRICE)} \]

Fig. 2

The relations representing the ERD in Fig. 1.
4. THE RRA OPERATORS AS MEANING OF NATURAL LANGUAGE CONSTRUCTS

The analogy between Natural Language and RRA is exposed when Entity sets and relationship sets in an ERD are expressed as relations in the most straightforward way:

For each entity set there is a relation whose attributes are these of the entities in the set;

For each relationship there is a relation whose attributes are these of the relationships in the set, augmented with the key attributes in all the entity sets associated by the relationship (key attributes are such that given its values, an entity is uniquely specified. Since a key in a relationship is used as an identifier only, we shall view all the key attributes of an entity appearing in a relationship as a simple attribute with values consisting of concatenating key attribute values of entities in a certain defined order).

The relational representation of the ERD of Fig. 1 appears in Fig. 2. If the DB relations are not such, it can sometimes (intuitively, if inherently the DB bears the information about the entities and the relationships. See [MMR]) be converted into this form using RRA operations. We shall not go into the conversion problem, and assume that the DB relations are in the desired form.

In what follows the Natural Language analogies of the RRA operations are given (For the NL combinations see [WIN]).

4.1 RELATIVIZATION AND THE NATURAL JOIN (PRODUCT) OPERATION

Relativization is connecting a sequence of sentences such that any two neighboring sentences are chained on an object part of the first one. One possibility is that the last object part of the first sentence is the subject of the second. For example:

SUPPLIER STOCKS ITEM REQUESTED by DEPARTMENT.

is a combination of SUPPLIER STOCKS ITEM and ITEM IS REQUESTED by DEPARTMENT.

The meaning is given by the RRA expression

STOCK*REQUEST.

where

STOCK=STOCK(SUPPLIER_key, ITEM_key, ...)
REQUEST=REQUEST(ITEM_key, DEPARTMENT_key, ...)
Each tuple in the resulting relation includes keys of a supplier, an item and a department such that the supplier really stock this item, and this item is really requested by this department. The embedded join in the product operation takes care that each tuple is constructed of tuples in STOCK and REQUEST which match on the ITEM’s key.

Suppose that we want to make the following query:

Get the SUPPLIERS who STOCK (any) ITEM REQUESTED by (any) DEPARTMENT.
The desired suppliers are received by performing the PROJECT operation by the SUPPLIER’s key on the above expression.

Now suppose that we change a little bit the query and ask:

Get the NAMES of SUPPLIERS who STOCK ITEM REQUESTED by DEPARTMENT.
Since NAME is an attribute of SUPPLIER and not of the relationship STOCK the projection by NAME should be performed on

"SUPPLIER*STOCK*REQUEST"

The first product operation has noting to do with the relativization, but is needed here to connect the supplier keys, which appear in STOCK with appropriate attribute names which appear in SUPPLIER. This kind of "gluing" an entity-relation with a relationship-relation using a natural-join appears also for other NL constructs, whenever an entity’s attribute appears explicitly in the construct.
4.2 RESTRICTION BY A CONSTANT AND THE SELECT OPERATION

A restriction is a construct where objects are characterized by a specific value or values of one of their attributes. A specific value may be defined as a constant.

For example:

The DEPARTMENT HAS the NAME (EQUALS to) "engineering". or

The ITEM HAS TYPE GREATER than (> 3). or

The SUPPLIER STOCKS QUANTITY of ITEMS SMALLER than 7.

The meaning is given by the following three expressions respectively:

\[ \text{select}_{\text{name}}=\text{ENGINEERING}(\text{DEPARTMENT}). \]
\[ \text{select}_{\text{type}} > 3(\text{ITEM}). \]
\[ \text{select}_{\text{quantity}} < 7(\text{STOCK}). \]

4.3 RESTRICTION BY A VARIABLE AND THE \( \theta \)-JOIN OPERATION

Restriction can appear also with a comparison of an attribute to an attribute (the same, or different) of another entity.

For example:

The EMPLOYEE HAS NAME (which is) EQUAL to NAME of SUPPLIER. or

The ITEM HAS TYPE GREATER than FLOOR OF DEPARTMENT.

The meaning is given by the following expressions respectively:

\[ \text{EMPLOYEE} \join_{\text{EMPLOYEE.NAME} = \text{SUPPLIER.NAME}} \text{SUPPLIER}. \]
\[ \text{ITEM} \join_{\text{TYPE} > \text{FLOOR}} \text{DEPARTMENT}. \]

REMARK: In the first example the attributes NAME should be renamed, because the operation is defined only for different such (see Sec. 2.7).
4.4 COORDINATION

Coordination means connecting sentences which have a common subject by the logical connectives "and", "or".

4.4.1 OR COORDINATION AND THE BORDERED UNION

For example

The ITEM IS STOCKED by SUPPLIER OR REQUESTED by DEPARTMENT.

which its meaning is given by

STOCK or REQUEST

The resulting relation includes tuples with keys of SUPPLIER, ITEM, DEPARTMENT such that either the item is stocked by the supplier or the item is requested by the department. If ITEM is stocked by SUPPLIER the combination ITEM, SUPPLIER appears with all the values of DEPARTMENT in REQUEST. Similarly, if ITEM is requested by DEPARTMENT, the combination ITEM, DEPARTMENT appears with all the values of SUPPLIER in STOCK.

4.4.2 AND COORDINATION AND THE BORDERED INTERSECTION (PRODUCT)

In the case of AND coordination the PRODUCT operator which reduces to BORDERED INTERSECTION gives the right meaning. For example, if in the example of section 4.4.1 we replace the OR by NOT, the appropriate RRA expression is

STOCK • REQUEST

which gives all tuples with ITEM, SUPPLIER, DEPARTMENT such that ITEM is both stocked by SUPPLIER and requested by DEPARTMENT.

4.5 NEGATION AND THE NOT (COMPLEMENT) OPERATION

When the word "NOT" appears in a sentence the meaning of the sentence turns to its logical complement. Assuming that the DB includes true facts and what is not included is not true (The closed world assumption [RE1]), the complement operator enables us to compute the opposite meaning of a sentence.
As we have seen, the meaning of

**SUPPLIER STOCKS ITEM**

is given by the relation STOCK. In STOCK all the facts about "who" stocks "what" are kept. The meaning of the opposite sentence

**The SUPPLIER DOES NOT STOCK ITEM**.

is all the relevant pairs of "who" and "what" which do not appear in STOCK. This is accepted by the following expression:

\[ \text{not(SUPPLIER(SUPPLIER-key),ITEM(ITEM-key))}(STOCK) \].

**REMARK**: When a NL expression is a combination of negation and coordination it can be converted using the De-Morgan formulas. This property is induced on the RRA for the NOT, BORDERED UNION and BORDERED INTERSECTION.

### 4.6 SET COMPARISON, UNIVERSAL QUANTIFIERS AND THE SET JOIN (GENERALIZED DIVISION)

Set comparison is not "natural" in NL, but is equivalent to universal quantifiers like "all", "at least", "more than" etc., which appear frequently. Suppose we want to check which supplier stocks all the items in the database. Such supplier can be characterized as follows:

**The SUPPLIER STOCKS a SET of ITEMS which CONTAINS the SET of (all) ITEMS**

The operation which gives all the suppliers satisfying this sentence is the SET-JOIN:

\[ \text{STOCK \text{set-join} ITEM} \]

Suppose now that we want to check which supplier stocks all the items that any other supplier supplies. Such supplier can be characterized as follows:

**The SUPPLIER STOCKS a SET of ITEMS which CONTAINS the SET of ITEMS SUPPLIED by (any) SUPPLIER**

The operation which gives all the suppliers satisfying this sentence is again the SET-JOIN:

\[ \text{STOCK \text{set-join} SUPPLY} \]
A special case of this construct is when the supplier stocks all the items that he supplies. Such supplier is characterized by the same RRA expression with renamed attributes in a certain way. As a consequence the division is automatically followed by a selection of tuples such that the first and the last supplier are the same. This issue called referencing is taken care of in Part 6.

4.7 AGGREGATE FUNCTIONS AND THE AF OPERATORS

Aggregate functions are related to operations performed on sets. In NL their names appear as common keywords like SUM, NUMBER OF (COUNT), MINIMUM, MAXIMUM, AVERAGE, etc. with a well defined meaning.

For example:

'The SUPPLIER SUPPLYING NUMBER (COUNT) of ITEMS...

(The three dots mean that usually there is a continuation of such construct, e.g., the SUPPLIER SUPPLYING NUMBER of ITEMS GREATER than 100.)

or

The MINIMAL QUANTITY for (any) ITEM STOCKED BY (any) SUPPLIER...

or

The DEPARTMENT REQUESTS SUM of QUANTITIES of ITEMS...

The respective AF operations are

\[
\begin{align*}
COUNT_{ITEM \rightarrow key} & (SUPPLY) \\
MIN_{QUANTITY} & (STOCK) \\
SUM_{QUANTITY} & (REQUEST)
\end{align*}
\]
4.8 SUMMARY

The meanings of NL constructs expressed by respective RRA operations are summarized in the following table.

<table>
<thead>
<tr>
<th>Examples</th>
<th>NATURAL LANGUAGE (Syntax)</th>
<th>RRA (Semantics)</th>
</tr>
</thead>
<tbody>
<tr>
<td>...ITEM HAVING COLOR = &quot;RED&quot;</td>
<td>Restriction by a constant</td>
<td>Select</td>
</tr>
<tr>
<td>...ITEM HAVING COLOR = COLOR OF ITEM...</td>
<td>Restriction by a variable</td>
<td>( \theta )-join (parameters: ( \neq, =, &gt;, &lt;, \geq, \leq ))</td>
</tr>
<tr>
<td>...ITEM REQUESTED BY DEPARTMENT MANAGED BY EMPLOYEE...</td>
<td>Relativization</td>
<td>Product (reduces to Natural-join)</td>
</tr>
<tr>
<td>...DEPARTMENT REQUESTING ITEM... AND MANAGED BY EMPLOYEE...</td>
<td>AND coordination</td>
<td>Product (reduces to intersection)</td>
</tr>
<tr>
<td>...DEPARTMENT EMPLOYING EMPLOYEE... OR HAVING LOCALITY...</td>
<td>OR coordination</td>
<td>Bordered union</td>
</tr>
<tr>
<td>...DEPARTMENT NOT REQUESTING...</td>
<td>Negation</td>
<td>Complement</td>
</tr>
<tr>
<td>...SET ITEM CONTAINS SET ITEM...</td>
<td>Set comparison (Universal Quantifiers)</td>
<td>Set-join (Generalized division; parameters: ( \neq, =, \subseteq, \subset, \supseteq, \supset ))</td>
</tr>
<tr>
<td>...SUM QUANTITY...</td>
<td>Aggregate function</td>
<td>AF: sum, min, max, count</td>
</tr>
</tbody>
</table>

The RRA RENAME and PROJECT do not have matching NL constructs but they are necessary: the first one to modify attribute names not taking part in an embedded join (see Part 6); the second - to drop unnecessary attributes from relations taking part in an RRA expression.
5. THE RRA EXPRESSIONS OF COMPLEX SENTENCES

It is assumed here that most NL sentences based on the information carried by a given ERD can be decomposed to the basic constructs described in Part 4, or are transformable to equivalent such. We shall demonstrate here an algorithm for constructing the RRA expression of such sentences.

The algorithm which views the NL sentence as a tree, can be executed by scanning it once, but will be described here as consisting of several steps for clarity. The description here is informal, and given through two examples, which cover most of its details and use all the basic constructs of Part 4.

Example 1: Suppose we want to construct the RRA expression of the following sentence:

"Departments requesting any item which is supplied by supplier located in HAIFA, to department which requests all the red colored items".

Step 1: TRANSFORMATION INTO BASIC ELEMENTS

This transformation takes off all the unnecessary elements in the sentence and leaves its skeleton only. The more sophisticated the transformation mechanism is, the more flexible NL constructs a system can handle. The methods for carrying out such transformations are out of this paper's scope but it is worthwhile noting that for the English subset which consists of the "surface sentences" combinations analog to the RRA operations only (which is sufficient from the expressive power point of view) this transformation is trivial. The skeleton consists of elements in the ERD derived from the NL construct and appear in the right order, and the minimal additional information needed to construct the RRA expression.

The transformed sentence is:

DEPARTMENT REQUEST ITEM SUPPLY SUPPLIER LOCATION = "HAIFA",
DEPARTMENT REQUEST SET ITEM CONTAINS SET ITEM COLOR = "RED".

This sentence (as the original one) describes a "walk" through the ERD with possible "jumps" in attribute and set comparisons, and also where there is a branching due to non binary relationship. The last kind of jump, due to SUPPLY, appears after the constant "HAIFA".
Step 2: RENAMING

Attribute renaming is performed while passing the sentence in its usual order. Entity-names are considered as entity-key attributes. The renaming is the following:

DEPARTMENT-key $\rightarrow x_1$
ITEM-key $\rightarrow x_2$
SUPPLIER-key $\rightarrow x_3$
LOCALITY $\rightarrow x_4$

DEPARTMENT-key $\rightarrow x_5$
ITEM-key $\rightarrow x_6$
ITEM-key $\rightarrow x_7$
COLOR $\rightarrow x_8$

Step 3: PROJECTION

Temporary relations are constructed from the renamed DB relations by taking projections by the renamed attributes only. A projection of a relationship-relation is taken whenever its name appears, and a projection of an entity relation - when its name appears together with one of its attribute names:

In the example it will be the following:

REQUEST $\rightarrow T_1(x_1,x_2)$
SUPPLY $\rightarrow T_2(x_2,x_3,x_5)$
SUPPLIER $\rightarrow T_3(x_3,x_4)$
ITEM $\rightarrow T_4(x_5,x_6)$

Step 4: BUILDING THE CONSTRUCT'S HYPER-TREE (HT)

We present the construct as a Hyper-Tree, built by following rules:

1. Each $x_i$ occurrence in any $T_j$ is a node.
2. Each constant is a node.
3. Each $T_j$ is a "fat" hyper-arc consists of its attribute's occurrences.
4. Attribute occurrences with the same name, which are representing the same element in the sentence are connected by a "thin" arc.
5. An attribute and a constant, two attributes or two sets with a comparison operation between them are connected with a "thin" arc.
The HT becomes directed by letting the first occurrence of the first attribute in the sentence to be the root.

The HT of the example appears in Fig. 3.

![Hyper-Tree Diagram]

**Fig. 3**
The Hyper-Tree of Example 1.

**Step 5: MARKING THE THIN ARCS WITH RRA OPERATORS**

Decomposing the transformed sentence into basic constructs it is easy to check that there is a natural one to one correspondence between the basic constructs and the thin arcs in the HT. Let us mark the thin arcs with RRA operators which match the corresponding basic constructs, as shown in Fig. 3.

**Step 6: CONSTRUCTING THE RRA EXPRESSION**

Actually the construction has been started in defining \( T_1 \) to \( T_5 \). It continues by collecting the RRA operators on the thin arcs of the HT, moving from the leaves to the root, using the neighboring "fat" arcs as operands. Hence, presenting the RRA expression as a sequence of (calculated) temporary relations the remaining ones are the following:
Example 2:

In this example do appear all the basic constructs of Part 4 that do not appear in the previous example. This implies the addition of some more types of elements to the Hyper-Tree.

The sentence is the following:

"Department located in floor lower than the floor of department requesting number of items greater than 20, or not supplied by supplier named TOM, and has the name ENGINEERING."

Step 1: TRANSFORMATION INTO BASIC ELEMENTS

This sentence is ambiguous and has five different meanings. Ambiguity resolution is an important part in the transformation problem, and can be carried through interaction with the user or by adding special symbols to the sentence. Here we shall choose arbitrarily one version and enforce it by putting parentheses in the transformed sentence.

\[
\text{DEPARTMENT} \left( \text{FLOOR} < \text{FLOOR DEPARTMENT REQUEST COUNT ITEM} > 20 \right) \\
\text{OR} \left( \text{NOT SUPPLY SUPPLIER NAME} = \text{"TOM"} \right) \\
\text{AND NAME} = \text{"ENGINEERING"})
\]

Steps 2,3: RENAMING AND PROJECTION

\[
\begin{align*}
\text{DEPARTMENT-key} & \rightarrow x_1 & \text{COUNT ITEM} & \rightarrow x_6 \\
\text{FLOOR} & \rightarrow x_2 & \text{SUPPLIER-key} & \rightarrow x_7 \\
\text{FLOOR} & \rightarrow x_3 & \text{NAME} & \rightarrow x_8 \\
\text{DEPARTMENT-key} & \rightarrow x_4 & \text{NAME} & \rightarrow x_9 \\
\text{ITEM-key} & \rightarrow x_5
\end{align*}
\]
DEPARTMENT → \( T_1(x_1, x_2) \)  
DEPARTMENT → \( T_2(x_3, x_4) \)  
REQUEST → \( T_3(x_4, x_5) \)  
SUPPLY → \( T_4(x_1, x_7) \)  
SUPPLIER → \( T_5(x_7, x_8) \)  
DEPARTMENT → \( T_6(x_1, x_9) \)  

Steps 4,5: CONSTRUCTING THE HYPER-TREE MARKED WITH OPERATORS

The OR and the AND operators bring new types of arcs to the HT. They are denoted graphically as triangles directing the operations' results towards the root. The triangles' vertices are nodes representing the same attribute.

Also the AF operators are denoted as triangles pointing to the node in a "fat" arc which represents the relation's attribute to be replaced by the AF (i.e., \( x_5 \)). The replacing attribute (i.e., \( x_6 \)) is represented by a node in the AF's triangle.

The example's HT appears in Fig 4.

![Hyper-Tree](image)

Fig. 4

The Hyper-Tree in example 2.
Step 6: THE RRA EXPRESSION

The AF operators differ from all other operators in the fact that they always should be evaluated before their neighbor operator in the direction of the HT's leaf (This causes a replacement of an attribute in the AF's operand represented by the "fat" arc neighboring the AF's triangle by the root's direction).

The RRA expression is represented by the renamings and projections in Steps 2,3 together with the following sequence of operations which is derived from the HT above.

\[
\begin{align*}
T_8(x_4,x_6) &= \text{COUNT}_{x_4}(T_3) \\
T_9(x_4) &= \text{select}_{x_0=20}(T_8) \\
T_{10}(x_3,x_4) &= T_2 * T_9 \\
T_{11}(x_1,x_4) &= T_1 \ominus \text{join}_{x_4} T_{10} \\
T_{12}(x_7) &= \text{select}_{x_7=\text{COMP}}(T_3) \\
T_{13}(x_1,x_7) &= T_4 * T_{12} \\
T_{14}(x_1,x_7) &= \text{not}_{x_1,x_7}(T_{13}) \\
T_{15}(x_1) &= \text{select}_{x_1=\text{ENGINEERING}}(T_6) \\
T_{16}(x_1,x_7) &= T_{14} * T_{13} \\
\text{RESULT} &= T_{17}(x_1,x_4,x_7) = T_{11} \text{ or } T_{16}
\end{align*}
\]

6. THE REFERENCE AND THE EMBEDDED JOIN

Referencing is common in NL. It is used in NL via expressions like "the same", "the-above", "that", etc. For example,

"Department requests item which is supplied to IT."

"to IT" means referring to the same department mentioned at the beginning of the sentence.

Computing tuples which satisfy such a sentence as a predicate, is equivalent to computing tuples which satisfy the following predicate:

"Department requests item which is supplied to department."

and then selecting only the tuples where the two departments' values are the same.

The RRA was designed to take care of references automatically while using the algorithm described in Part 5. It is done by a natural join operation embedded in the RRA's operators (see definitions in Part 2). All that should be done is to rename the entities involved in referencing with the same name. Since the embedded join is automatically performed on attributes with the same names, the desired result is achieved.

Applying the algorithm of Part 5 to the example above is as follows:
Step 1: TRANSFORMATION TO BASIC ELEMENTS

DEPARTMENT(X) REQUEST ITEM SUPPLY DEPARTMENT(X).

The symbol X (referencing symbol) indicates that the marked identities are involved in referencing.

Step 2: RENAMING

DEPARTMENT(X)-key → x₁
ITEM-key → x₂
DEPARTMENT(X)-key → x₁

Because of the reference the two occurrences of DEPARTMENT have the same renaming.

Steps 3-6: PROJECTING AND GETTING THE RRA EXPRESSION

REQUEST → T₁(x₁,x₂)
SUPPLY → T₂(x₂,x₁)

The RRA expression is

RESULT = T₁ * T₂

and the product operation reduces to a natural join over x₁ and x₂.
7. QUERIES AND DATABASE MANIPULATIONS

Till now we have dealt with computing a relation which tuples consist of attributes’ values satisfying a predicate expressed in NL. Phrasing a query, we are usually interested in values of additional or different attributes connected with the entities and relationships which appear in the predicate. For this purpose we can use a mechanism which already exists in our method - the referencing mechanism.

For example,

"get name of department and name of item such that THIS item is requested by the ABOVE department."

Usually there is a clear separation between the list of requested attributes (the request list) and the predicate, like in this example by the word "such". It is convenient to keep this separation in the transformed sentence which can be (depending on specific implementation) the following:

\[
\text{GET - NAME DEPARTMENT}(X), \text{NAME ITEM}(Y) - \text{ITEM}(Y) \text{ REQUEST DEPARTMENT}(X)
\]

All that is left to do is to continue with the usual procedure, and to join (by PRODUCT) at the end all the temporaries which are derived from the request list together with the temporary which is the result of the predicate.

Also other manipulations can be carried using the predicate part as an instrument for identifying entities or relationships (finding values of keys) to be modified.

For example,

"Change to blue the color of all red items"

The transformed sentence can be the following:

\[
\text{CHANGE - ITEM}(X) \text{ COLOR}="\text{BLUE}" - \text{ITEM}(X) \text{ COLOR}="\text{RED}"
\]
8. CONCLUSION

A simple methodology for manipulating relational databases by NL subsets and NL-like languages was presented. The methodology is based on RRA whose operators give the meaning of the basic NL constructs. The direct, simple translation to operations upon the database enables the construction of efficient interfaces. The transformation part, which is not described here, can be carried out easily for fragments of NL, which have well defined syntax, or can be analyzed by other methods ([BF],[WIN]). The same is applicable for formal NL-like languages, which have a closed description..

The transformed sentences have a simple BNF syntax (not given here), and the entire algorithm can be executed while parsing. Thus the transformation part is usually not necessary for languages which can be parsed.

The methodology's simplicity enables it to be implemented as domain independent (external surface sentences definition) system on contemporary personal computers. The methodology has been developed and implemented during the ERROL project. ERROL (an Entity Relationship Role Oriented Language) is an English like language. The project has focused on user interface to relational databases based on ERROL ([ALP],[COH],[RCI],[IR]). The examples in parts 5,6,7 as expressed and executed in the ERROL System appear in the Appendix.

ACKNOWLEDGEMENT

Thanks due to my graduate students, especially Victor M. Markowitz and Reuven Cohen, with whom many ideas in this paper have been shaped and tested.
REFERENCES


APPENDIX

The methodology described in this paper has been implemented in the ERROL System (ERROL - an Entity Relationship Role Oriented Language - [MAR],[MR1],[IR]), which is a DBMS over the Entity-Relationship Model. ERROL is an English-like query language which has the basic syntactical constructs described in Part 4 of the paper. The language is extendible in several aspects by a rule base subsystem which can transform other syntactical constructs to the basic ones.

Examples 1,2 of Part 5 in the paper and the examples of Parts 6,7 are demonstrated here via the ERROL System's sessions in examples 1,2,3,4 respectively. The examples are phrased using the basic constructs of ERROL in order to emphasize the direct mapping to RRA. For each example the compiler output and the query computation are displayed. The ERROL to RRA compiler output consists of a sequence of RRA operations, a renaming table (correlation symbols) and a table of the projected relations (operational scheme).

REMARKS:

(1) The temporary relations' indexings in the paper's body and here are different since the ERROL translation is performed in one step.

(2) The parenthesis in example 2 are necessary in ERROL to resolve coordination (branching) ambiguities. It indicates that the coordination is on the first occurrence of department (and not on the second and not on supplier). The logical operators' precedence is "not", "and" and "or" as common in programming languages.

(3) In example 4 the values of elements in the request list are not computed by Natural Join operations as suggested in Part 7., but are kept by the ERROL system from the initial projections.
Example 1

get department requesting item supplied by supplier having
locality = "haifa" to department requesting set item contains
set item having color = "red"

The sequence of operators :
-----------------------------
project(t3 , supplier)
select(t4 , t3 , x5 , "haifa", =)
project(t2 , supply)
product(t5 , t4 , t2)
project(t7 , t6 , "red", =)
project(t6 , request)
divide(t5 , t6 , t7 , x5 , x6 , contains)
product(t10 , t9 , t5)
project(t1 , request)
project(t11 , t10 , t1)
print_on(t11 , _ t1)

The correlation symbols :
-----------------------
<table>
<thead>
<tr>
<th>c_symbol</th>
<th>org_name</th>
<th>c_counter</th>
<th>op_ref</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>department</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>x1</td>
<td>item</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>x2</td>
<td>supplier</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>x3</td>
<td>locality</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>x4</td>
<td>Department</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>x5</td>
<td>item</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>x6</td>
<td>item</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>x7</td>
<td>color</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

The operational scheme :
------------------------
<table>
<thead>
<tr>
<th>valid</th>
<th>counter</th>
<th>c_symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>1</td>
<td>x0 x1</td>
</tr>
<tr>
<td>t2</td>
<td>1</td>
<td>x1 x2 x4</td>
</tr>
<tr>
<td>t3</td>
<td>1</td>
<td>x0 x2 x4</td>
</tr>
<tr>
<td>t4</td>
<td>1</td>
<td>x2 x4 x7</td>
</tr>
</tbody>
</table>

supplier entity

<table>
<thead>
<tr>
<th>name</th>
<th>city</th>
<th>locality</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>11111</td>
<td>haifa</td>
</tr>
</tbody>
</table>

item entity

<table>
<thead>
<tr>
<th>name</th>
<th>color</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345</td>
<td>blue</td>
<td>23456</td>
</tr>
<tr>
<td>12346</td>
<td>red</td>
<td>23456</td>
</tr>
</tbody>
</table>

request relationship
----------------------
<table>
<thead>
<tr>
<th>sno</th>
<th>lino</th>
<th>lino</th>
<th>quantity</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111</td>
<td>12345</td>
<td>23456</td>
<td>100</td>
<td>12345</td>
</tr>
<tr>
<td>12345</td>
<td>12346</td>
<td>23456</td>
<td>100</td>
<td>12345</td>
</tr>
<tr>
<td>13333</td>
<td>12345</td>
<td>23456</td>
<td>100</td>
<td>12345</td>
</tr>
<tr>
<td>12345</td>
<td>67890</td>
<td>23456</td>
<td>100</td>
<td>12345</td>
</tr>
</tbody>
</table>

Example 2

get department requesting item supplied by supplier having
locality = "haifa" to department requesting set item contains
set item having color = "red"

The sequence of operators :
-----------------------------
project(t3 , supplier)
select(t4 , t3 , x5 , "haifa", =)
project(t2 , supply)
product(t5 , t4 , t2)
project(t7 , t6 , "red", =)
project(t6 , request)
divide(t5 , t6 , t7 , x5 , x6 , contains)
product(t10 , t9 , t5)
project(t1 , request)
project(t11 , t10 , t1)
print_on(t11 , _ t1)
Example 2

```c
void get_department ( having floor ( floor of department requesting count item ) 20 )
    
// or not supplied by supplier having name = "tom"

// and having name = "engineering"

/***** COMPILER OUTPUT *******/

The sequence of operators:

- project(t4 , request)
- count(t5 , t4 , x5 , x4)
- select(t5 , t5 , x5 , x9 , )
- project(t3 , department)
- product(t7 , t6 , t3)
- project(t2 , department)
- t_join(t8 , t2 , t1 , x1 , x2 , )
- project(t10 , supplier)
- select(t11 , t10 , x7 , "tom" , )
- project(t9 , supply)
- product(t12 , t11 , t9)
- nat(t13 , t12)
- project(t14 , department)
- select(t15 , t14 , x8 , "engineering" , )
- product(t16 , t13 , t15)
- or(t17 , t9 , t16)
- pr in(l17 , nil)

The correlation symbols:

<table>
<thead>
<tr>
<th>c_symbol</th>
<th>org_name</th>
<th>c_counter</th>
<th>op_ref</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>department</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>x1</td>
<td>floor</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>x2</td>
<td>floor</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>x3</td>
<td>department</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>x4</td>
<td>item</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>x5</td>
<td>cnt_item</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>x6</td>
<td>supplier</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>x7</td>
<td>name</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>x8</td>
<td>name</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

The operational scheme:

<table>
<thead>
<tr>
<th>rmid</th>
<th>counter</th>
<th>c_symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>t2</td>
<td>1</td>
<td>x0 x1</td>
</tr>
<tr>
<td>t3</td>
<td>1</td>
<td>x2 x3</td>
</tr>
<tr>
<td>t4</td>
<td>1</td>
<td>x3 x4</td>
</tr>
<tr>
<td>t5</td>
<td>1</td>
<td>x0 x9</td>
</tr>
<tr>
<td>t10</td>
<td>1</td>
<td>x6 x7</td>
</tr>
<tr>
<td>t10</td>
<td>1</td>
<td>x0 x8</td>
</tr>
</tbody>
</table>

// request relationship

<table>
<thead>
<tr>
<th>dno</th>
<th>floor</th>
<th>name</th>
<th>quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111</td>
<td>12345</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>11111</td>
<td>12345</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>12222</td>
<td>12345</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>12222</td>
<td>12346</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>33333</td>
<td>12346</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>33333</td>
<td>12346</td>
<td>S1</td>
<td>1</td>
</tr>
</tbody>
</table>

// t3 relation

<table>
<thead>
<tr>
<th>dno</th>
<th>floor</th>
<th>name</th>
<th>quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111</td>
<td>12345</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>11111</td>
<td>12346</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>12222</td>
<td>12345</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>12222</td>
<td>12346</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>33333</td>
<td>12346</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>33333</td>
<td>12346</td>
<td>S1</td>
<td>1</td>
</tr>
</tbody>
</table>

// t4 relation

<table>
<thead>
<tr>
<th>dno</th>
<th>floor</th>
<th>name</th>
<th>quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111</td>
<td>12345</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>11111</td>
<td>12345</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>12222</td>
<td>12345</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>12222</td>
<td>12346</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>33333</td>
<td>12345</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>33333</td>
<td>12346</td>
<td>S1</td>
<td>1</td>
</tr>
</tbody>
</table>

// t5 relation

<table>
<thead>
<tr>
<th>dno</th>
<th>floor</th>
<th>name</th>
<th>quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111</td>
<td>12345</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>11111</td>
<td>12346</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>12222</td>
<td>12345</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>12222</td>
<td>12346</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>33333</td>
<td>12345</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>33333</td>
<td>12346</td>
<td>S1</td>
<td>1</td>
</tr>
</tbody>
</table>

// t6 relation

<table>
<thead>
<tr>
<th>dno</th>
<th>floor</th>
<th>name</th>
<th>quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111</td>
<td>12345</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>11111</td>
<td>12346</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>12222</td>
<td>12345</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>12222</td>
<td>12346</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>33333</td>
<td>12345</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>33333</td>
<td>12346</td>
<td>S1</td>
<td>1</td>
</tr>
</tbody>
</table>

// t7 relation

<table>
<thead>
<tr>
<th>dno</th>
<th>floor</th>
<th>name</th>
<th>quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111</td>
<td>12345</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>11111</td>
<td>12346</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>12222</td>
<td>12345</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>12222</td>
<td>12346</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>33333</td>
<td>12345</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>33333</td>
<td>12346</td>
<td>S1</td>
<td>1</td>
</tr>
</tbody>
</table>

// t8 relation

<table>
<thead>
<tr>
<th>dno</th>
<th>floor</th>
<th>name</th>
<th>quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111</td>
<td>12345</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>11111</td>
<td>12346</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>12222</td>
<td>12345</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>12222</td>
<td>12346</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>33333</td>
<td>12345</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>33333</td>
<td>12346</td>
<td>S1</td>
<td>1</td>
</tr>
</tbody>
</table>
```

Technion - Computer Science Department - Technical Report CS0410 - 1986
Example 3

> get department(x) requesting item supplied to department(x)

> /cat

************************** COMPILER OUTPUT **************************

The sequence of operators:

project(t2, supply)
project(t1, request)
print_int(t3, nil)

The correlation symbols:

c_symbol org_name c_counter op_ref

| x0 | department 3 | x1 | item 2 |

The operational scheme:

<table>
<thead>
<tr>
<th>relid</th>
<th>counter</th>
<th>c_symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>1</td>
<td>x0 x1</td>
</tr>
<tr>
<td>t2</td>
<td>1</td>
<td>x1 x0</td>
</tr>
</tbody>
</table>

> /xt

supply relationship

<table>
<thead>
<tr>
<th>lno</th>
<th>lno</th>
<th>idno</th>
<th>quantity</th>
<th>price</th>
</tr>
</thead>
</table>

> /xt

supply relationship

<table>
<thead>
<tr>
<th>lno</th>
<th>lno</th>
<th>idno</th>
<th>quantity</th>
<th>price</th>
</tr>
</thead>
</table>

Example 4

```sql
> get name of department(x), name of item requested by department(x)
> set

********** COMPILER OUTPUT **********

The sequence of operators:
-------------------------
project(t1, department)  
project(t2, request)    
product(t4, t3, t2)     
print_in(t5, nil)

The correlation symbols:
-------------------------
c_symbol  org_name  c_counter  op_ref
----------  --------  ---------  --------
x0       name      2          0
x1       department 2          0
x2       name      2          0
x3       item      3          6

The operational scheme:
-------------------------
relid  counter  c_symbols
----------  ---------  ----------------------
t1       1          x0 x1
x1 x2 x3
x3 x1

) / <

department entity
----------------
<table>
<thead>
<tr>
<th>dno</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>111111</td>
<td>manpower</td>
</tr>
<tr>
<td>122222</td>
<td>production</td>
</tr>
<tr>
<td>333333</td>
<td>engineering</td>
</tr>
</tbody>
</table>

t1 relation
----------------
| x0 | x1 |
| engineering | 333333 |
| manpower | 111111 |
| production | 222222 |

request relationship
----------------
<table>
<thead>
<tr>
<th>dno</th>
<th>line</th>
<th>quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>111111</td>
<td>12345</td>
<td>5</td>
</tr>
<tr>
<td>111111</td>
<td>12346</td>
<td>5</td>
</tr>
<tr>
<td>122222</td>
<td>12345</td>
<td>5</td>
</tr>
<tr>
<td>122222</td>
<td>12346</td>
<td>5</td>
</tr>
<tr>
<td>333333</td>
<td>12345</td>
<td>5</td>
</tr>
</tbody>
</table>

t4 relation
----------------
| x1 | x2 |
| 111111 | auto |
| 111111 | bolt |
| 122222 | auto |
| 122222 | bolt |
| 333333 | auto |
| 333333 | bolt |

t5 relation
----------------
| lx3 | lx1 |
| 12345 | 11111 |
| 12345 | 22222 |
| 12345 | 33333 |
| 12346 | 11111 |
| 12346 | 22222 |
| 12346 | 33333 |

item entity
----------------
<table>
<thead>
<tr>
<th>line</th>
<th>name</th>
<th>color</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345</td>
<td>auto</td>
<td>red</td>
<td>A</td>
</tr>
<tr>
<td>12346</td>
<td>bolt</td>
<td>blue</td>
<td>B</td>
</tr>
<tr>
<td>12347</td>
<td>screw</td>
<td>green</td>
<td>B</td>
</tr>
</tbody>
</table>

t2 relation
----------------
| lx2 | lx3 |
| 12345 | auto |
| 12346 | bolt |
| 12347 | screw |
```

Technion - Computer Science Department - Technical Report CS0410 - 1986