THE ERROL SYSTEM

by

Ofer Imanuel and Yoav Raz

Technical Report #411

April 1986
The ERROL System

Ofer Immanuel
Yoav Raz
Technion, Israel

ABSTRACT

The ERROL system (ES) is an interactive multiuser entity-relationship database management system. ERROL - an Entity-Relationship Role Oriented Language - is an English-like query, data manipulation and data definition language. The main features of the system are user-friendliness, flexibility in complex data manipulations and minimal overhead in maintenance. ES uses relations to represent both entity sets and relationship sets. The relations are manipulated using RRA - a Reshaped Relational Algebra, which is an intermediate language in ERROL compilation. Being a multiuser system, ES has concurrency control and recovery mechanisms. ES was programmed in C, runs on a VAX machine under the UNIX operating system.
1. INTRODUCTION

The ERROL system (ES) is an interactive multiuser entity-relationship database management system. ERROL - an Entity-Relationship Role Oriented Language - is an English-like query, data manipulation and data definition language (see [MAR], [MR1], [RCM]).

An intermediate step in developing ES was constructing a prototype based on the INGRES DBMS (see [ALP]).

The main features of the system are user-friendliness, flexibility in complex data manipulations and minimal overhead in maintenance.

ES uses relations to represent both entity sets and relationship sets. The relations are manipulated using RRA - a Reshaped Relational Algebra, which is an intermediate language in ERROL compilation (see [RA1], [MR2], [MR3]).

Being a multiuser system, ES has concurrency control and recovery mechanisms (see [NAV], [MON]). ES was programmed in C (see [KER]), runs on a VAX machine under the UNIX operating system (see [UNI]).

Section 2 gives a short review of the ERM, its definition and linguistic aspects. In section 3 the ERROL language is described through many examples.

Section 4 describes the translation of ERROL. It is composed of two parts: the compilation from ERROL into RRA, and the RRA expression interpretation or its translation to C.

In section 5 the system architecture is examined. After a general introduction, the description of the system's five main parts is given: the SDBMS, the compiler from ERROL into RRA, the interpreter/compiler from RRA to C, the user's monitor, and the full screen entity editor.

Finally, section 6 reviews the engineering considerations, conclusions, and recommendations for continuing the work.

Two appendices are included: Appendix A is the manual of the ES system; Appendix B is a sample session.
2. THE ENTITY RELATIONSHIP MODEL (ERM)

2.1. DEFINITION OF THE ERM

The ERM views an enterprise as consisting of entities and relationships.

An entity is part of the enterprise (object, material or abstract), which can be distinguished from its environment and has a unique identification.

Entities are grouped into entity-sets (e-sets), which have unique names and types. Within an entity set all entities are described by the same attributes.

Example:

Department No. 4 is an entity in an organization. The set of departments is an entity set, named "DEPARTMENT". "DEPARTMENT NAME" is an attribute.

A relationship describes an interaction among entities. Relationships are elements of relationship-sets (r-sets). Each relationship is identified by the role it plays among entities and by the entities participating in it.

Relationships may have attributes which describe them.

Example:

Supplier 7 supplies part 2 to department 4 in quantity 28. This relationship belongs to the relationship-set "SUPPLY" and identified by its entities: Supplier 7, part 2, Department 4.

"QUANTITY" is an attribute of such relationship and its value for this specific one is 28.

An ERM schema (ERS) is a description of all entity, relationship and attribute types. A schema can be represented by a diagram appearing in figure 2.1.

2.2. THE LINGUISTIC ASPECT OF THE ERM

Every relationship type in the schema can be described by a simple sentence with a subject, verb and objects.

Example:

The relationship "SUPPLY" can be described by the following simple sentences.

Supplier supplying item to department. Item is supplied to department by supplier. Department is supplied with item by supplier.

In each sentence the verb part is called the role of the subject entity in the relationship.

An interaction between an entity or a relationship and its attribute can be described by a simple grammatical structure as well.

Example:

Name of department... Department having name...

These simple structures can be combined into compound sentences viewed as predicates for defining sets of data in the database.
Figure 2.

The Entity Relationship Diagram of an enterprise schema example.
Example:

Departments supplied by supplier having locality equal to the locality of supplier having number = "4", and managed by employee having salary greater than 1000.

ERROL takes advantage of the possibilities posed by the linguistic analogies of the ERM, by using denotations based on such simple sentences, and by using constructs similar to the natural sentence combination.
3. THE ERROL LANGUAGE

ERROL enables one to perform data manipulations required in a database based on the ERM, e.g., entity set and relationship set definition and its attribute characterization; insertion or deletion of entities and relationships; attribute values updating; retrieving information by using queries.

A predicate phrased in a natural language like structure may be associated to every operation. The operation is performed only for the data satisfying the predicate.

The expressing power of ERROL is at least as that of relational algebra and relational calculus ([ULL]). ERROL syntax is described in [RCI], [COH], and the essence of its semantics, using RRA, in [MR3].

The ERROL language is introduced here by a discussion of its basic constructs, and through a set of characteristic examples, which are based on the schema in Figure 1.

3.1. Queries

E-set names, attribute names and roles are the ERROL identifiers.

Most of these identifiers are declared in the Entity Relationship Scheme (ERS) over which the queries are expressed. ERROL queries may involve constants (numeric constants, e.g., 2, or strings, e.g., 'ABC').

A restriction is an elementary statement of the form "A \( \in \) c" or "A \( \notin \) B", where c is a constant, A and B are entities or attributes, and \( \in \) is a comparison operator.

ERROL accepts prepositions that may be placed within a query. Prepositions are useful in making a query look more natural, and thereby self-explanatory. For example:

SUPPLIER SUPPLYING ITEM IN QUANTITY TO DEPARTMENT

In the following presentation the ERROL key words are underlined.

3.1.1. The GET-CLAUSE

The target elements of a query are stated in the GET-CLAUSE, which is headed by the key-word GET and consists of a list of one or more sublists separated by semicolons. A sublist may contain:

(a) an entity - implying the request for the whole entity-representation for instance:

GET SUPPLIER;

(b) several attributes belonging to an entity, possibly followed by the roles corresponding to the respective properties, separated by commas: the last attributed is part of a full property denotation, including the entity implying the request for a partial entity-representation, for instance:

GET SALARY, NAME OF EMPLOYEE;

* This section follows [RCM].
(c) several attributes belonging to an r-set, appearing in an embedded property-denotation of properties belonging to an r-set - implying the request for a partial (or whole, if all the attributes are included) relationship-representation, for instance:

```
GET QUANTITY ITEM BY SUPPLIER TO DEPARTMENT;
```

Every entity of a sublist (a) or (b), and at least one entity of a sublist (c), will participate in the qualification phrase (TIS-CLAUSE; see below). If no qualification exists, and there is more than one sublist in the GET-CLAUSE, every sublist is considered as a separate request. The order of the sublists in the GET-CLAUSE is irrelevant.

A query statement always expresses a derived r-set defined on the e-sets appearing in the GET-CLAUSE sublists.

### 3.1.2. The TIS-CLAUSE

A GET-CLAUSE generally has, beside the target list, a statement asserting the required associations connecting the target e-sets. This statement is contained in a TIS-CLAUSE (TIS is the abbreviation of "THAT IS"). A TIS-CLAUSE is delimited by the key-word TIS and a semicolon. It is based on the simple association denotations of the ERS combined in a complex qualification phrase. The last semicolon/string of semicolons in a query statement is/are replaced by a period. In many cases the TIS keyword has not to be explicitly included in the query.

### 3.1.3. Referencing

Referencing is used when we refer in a query to an entity which has been mentioned before in the query. This is done by suffixing the entity with reference symbols which have to start with a "!" sign. For example:

```
DEPARTMENT!D, EMPLOYEE!E.
```

Such a referencing does not have the variants we find in the natural language, but is compact and unambiguous. In addition, the following implicit referencing can be used, when possible (no danger of ambiguity) to make it more compact:

(a) referenced e-set in the target list of a GET-CLAUSE should not be marked by "!x".

(b) an entity suffixed by a "!" without a symbol in the TIS-CLAUSE refers to the corresponding entity in the target list of the GET-CLAUSE.

### 3.1.4. Chaining

Chaining is the simplest way of connecting simple (explicit) associations. A certain denotation of an association reflects how in the corresponding paraphrases of the description sentence, an e-set is in the subject position, while all the other e-sets are in the object positions. The last entity of the GET-CLAUSE is the subject in the first association-denotation of the TIS-CLAUSE. Chaining is based on the natural language sentence combination by relativization. A first query example will illustrate the chaining of two r-set denotations.

*(ERROL 1)*

"Find the departments requesting some item stocked by some supplier"

```
GET DEPARTMENT REQUESTING ITEM STOCKED BY SUPPLIER.
```
Restrictions always appear chained with association denotations. For instance:

(ERROL 2)
"Find the suppliers supplying items of type 1"
GET SUPPLIER SUPPLYING ITEM HAVING TYPE = '1'.

A longer chaining is realized when the place of the constant is taken in the restriction by another attribute, belonging to some property.

(ERROL 3)
"Find the employees working in departments requesting items stocked by suppliers located in the same locations as those supplying red items:
GET EMPLOYEE EMPLOYED IN DEPARTMENT REQUESTING ITEM STOCKED BY SUPPLIER HAVING LOCALITY = LOCALITY OF SUPPLIER SUPPLYING ITEM HAVING COLOR = 'RED'.

In this example, notice that the different appearances of SUPPLIER do not refer to the same SUPPLIER since no correlation is mentioned.

(ERROL 4)
"Get the employees that earn more than their manager"
GET EMPLOYEE EMPLOYED BY DEPARTMENT MANAGED BY EMPLOYEE HAVING SALARY < SALARY OF EMPLOYEE!.

3.1.5. Negation
Any r-set, R, has a complementary r-set, that groups the "possible relationships" that are not elements of R. All the "possible" relationships of some r-set R are obtained by the Cartesian product of the e-sets on which the r-set is defined. The complementary r-set of an r-set R is denoted by the negated denotation of R.

(ERROL 5)
"Get all the items and the departments that do not request them"
GET ITEM; DEPARTMENT NOT REQUESTING ITEM!.

Similarly, the complementary of a complex (derived) r-set may be denoted by the negation of the chain expressing the r-set.

(ERROL 6)
"Get the pairs of suppliers and departments, such that the department is not requesting items stocked by the respective supplier"
GET SUPPLIER; DEPARTMENT NOT REQUESTING ITEM STOCKED BY SUPPLIER.

3.1.6. Branching
In addition to chaining, association denotations, whether simple or complex, may be connected by branching. Branching is based on the natural language sentence combination by coordination. The coordination connectors are AND and OR, while the coordination "pivot" is the entity or attribute in the subject position in all the coordinated denotations. In order to enforce a certain order of evaluation brackets may be used.

(ERROL 7)
"Get the departments requesting items that are stocked by suppliers from LONDON, and managed by managers earning more than 1000"
GET DEPARTMENT; ITEM TIS REQUESTING ITEM STOCKED BY SUPPLIER HAVING LOCALITY = 'LONDON' AND MANAGED BY EMPLOYEE HAVING SALARY >
3.1.7. Nesting

A branching may have as pivot any inner element of a TIS-CLAUSE, not necessarily the leading subject. In this case the respective element will head an inner TIS-CLAUSE.

(ERROL 9)

"Get the departments requesting item that are stocked by suppliers located in LONDON and having type T"

GET DEPARTMENT REQUESTING ITEM TIS STOCKED BY SUPPLIER LOCATED IN LOCALITY = 'LONDON' AND HAVING TYPE = 'T'.

An inner TIS-CLAUSE contains chains starting with roles corresponding to the pivot subject (e-set in the above example), like the general TIS-CLAUSE; inner TIS-CLAUSE may qualify attributes or sets as well.

(ERROL 10)

"Get the items that are red or blue"

GET ITEM HAVING COLOR TIS = 'RED' OR = 'BLUE'.

When the pivot is the subject of a restriction, the respective TIS-CLAUSE contains chains starting with comparison operators.

(ERROL 11)

"Get the items that are red or are colored like the items supplied by LONDON located suppliers"

GET ITEM HAVING COLOR TIS = 'RED' OR = COLOR OF ITEM SUPPLIED BY SUPPLIER HAVING LOCALITY = 'LONDON'.

It has been pointed out that chaining relies on the first and last elements of the association-denotation. When the association is n-ary (n > 2) and the qualification of one of the other elements is required, inner TIS-CLAUSE are used.

(ERROL 12)

"Get the departments requesting items supplied to them by suppliers located in PARIS, and placed on the 2-nd floor"

GET DEPARTMENT REQUESTING ITEM SUPPLIED BY SUPPLIER TIS HAVING LOCALITY = 'PARIS'; TO DEPARTMENT! HAVING FLOOR = 2.

3.1.8. Set Expressions

In ERROL the universal quantification is expressed by set expressions

(ERROL 13)

"Get the suppliers stocking at least all the items requested by some department"

GET SUPPLIER STOCKING SET ITEM CONTAINS SET ITEM REQUESTED BY DEPARTMENT.

Sets may be compared by set operators (EQ, IN, CONTAINS, NOT EQ, NOT IN, NOT CONTAINS), with other compatible sets, i.e., sets containing entities of a same type (e-set).

The set-expressions are a convenient way of expressing the universal quantifier. For simple queries, the only effort required from the user is to reformulate expressions involving "all", "at least", "at most", etc., by using
sets and set instances corresponding to a property. The set-comparisons then involve compatible sets containing values from a same value-set.

(ERROL 14)
“Get the pairs of departments and suppliers, such that a department requests items at least in quantity samples in which the supplier stocks them.”
GET SUPPLIER, DEPARTMENT REQUESTING SET QUANTITY CONTAINS SET QUANTITY STOCKED BY SUPPLIER.

A set-expression may not address all the e-sets on which an r-set is defined. Attention must be paid, in such cases, to the intended meaning. For instance, the following two queries clearly differ.

(ERROL 15)
“Get the items that are supplied by all the suppliers” (i.e., no matter to what department)
GET ITEM SUPPLIED BY SET SUPPLIER EQ SET SUPPLIER.

(ERROL 16)
“Get the items that are supplied to some departments, by all the suppliers”
GET ITEM SUPPLIED TO DEPARTMENT BY SET SUPPLIER EQ SET SUPPLIER.

Any entity of the set-expression can be qualified.

(ERROL 17)
“Get the departments supplied at least by Saxon and Peugeot suppliers”
GET DEPARTMENT SUPPLIED BY SET SUPPLIER CONTAINS SET SUPPLIER HAVING NAME TIS = 'SAXON' OR =

The set part may also be qualified, provided all the chains in the respective TIS-CLAUSE are headed by set comparison-operators.

(ERROL 18)
“Get the departments requesting at least all the red items plus all the items stocked by supplier 500”
GET DEPARTMENT REQUESTING SET ITEM TIS CONTAINS SET ITEM TIS HAVING COLOR = 'RED'; AND CONTAINS SET ITEM STOCKED BY SUPPLIER HAVING S# = '500'.

Set nesting is essential for the formulation of more complex universal-quantification queries. The natural language phrase expressing them has a recursive form, which the ERROL query reproduces. Hence the difficulty of formulating the query in ERROL parallels the difficulty of the corresponding natural language expression.

(ERROL 19)
“Get the departments that are supplied by all the suppliers supplying the department all the items they are stocking”
GET DEPARTMENT SUPPLIED BY SET SUPPLIER CONTAINS SET SUPPLIER TIS SUPPLYING DEPARTMENT! WITH SET ITEM CONTAINS SET ITEM TIS STOCKED BY SUPPLIER.

3.1.9. Aggregate Functions
In ERROL it is possible to express derived properties with the help of so-called aggregate functions (af): COUNT, SUM, MAX, MIN.

COUNT is applied directly to sets of entities and returns the number of entities
in these sets. The value-set associated with the new attribute is the set of the positive integer numbers:

(ERROL 20)

"Get the items that are stocked by at least 3 suppliers"

GET ITEM STOCKED BY COUNT SUPPLIER \( \geq 3 \).

SUM processes attribute-instances corresponding to a property, such that it takes into account any (duplicate) attribute-instances. The value-set associated with the new attribute is the same as the value-set of the attribute corresponding to the above property.

(ERROL 21)

"Get the departments requesting a total quantity of items greater than 22000"

GET DEPARTMENT REQUESTING SUM QUANTITY > 22000.

MAX and MIN are similar to SUM.

(ERROL 22)

"Get the departments whose minimal quantitative request are no less than 1000"

GET DEPARTMENT REQUESTING MIN QUANTITY \( \geq 1000 \).

A derived property, precisely as an explicit one, may be chained with other properties, whether they are derived or not.

(ERROL 23)

"Get the items requested in a total quantity equal to the quantity in which some supplier stocks them"

GET ITEM REQUESTED IN SUM QUANTITY = QUANTITY STOCKED ITEM! BY SUPPLIER.

A new attribute may be involved in restrictions or may be qualified by a TIS_CLAUSE, the same as an explicit attribute.

(ERROL 24)

"Get the items supplied by at least five suppliers and by no more than the number of all the suppliers supplying red items"

GET ITEM SUPPLIED BY COUNT SUPPLIER \( \geq 5 \) AND \( \leq \) COUNT SUPPLIER SUPPLYING ITEM HAVING 'COLOR' = 'RED'.

3.1.10. Creating an Entity-Set

(ERROL 25)

DEFINE ENTITY ITEM KEY ID = 15,
TYPE = C2, COLOR = C10, NAME = C10.

3.1.11. Creating a Relationship-Set

(ERROL 26)

DEFINE RELSHIP SUPPLY, QUANTITY = 15, PRICE = 110 BETWEEN ITEM SUPPLIED, SUPPLIER SUPPLYING, DEPARTMENT SUPPLIED.

3.1.12. Destroying an Entity-Set or a Relationship-Set

Remark: The destruction of an entity-set is implying the destruction of all relationship-sets in which the entity-set is participating.

(ERROL 27)

DESTROY ITEM.
3.1.13. Inserting an Entity
(ERROL 28)
\[
\text{INSERT ITEM'COLOR = 'RED', I# = 53.}
\]

3.1.14. Inserting a Relationship
(ERROL 29)
\[
\text{INSERT SUPPLY QUANTITY = 100, PRICE = 50 WHERE ITEM HAVING I# = 55, SUPPLIER SUPPLYING ITEM, DEPARTMENT TIS' HAVING FLOOR = 5 OR MANAGED BY EMPLOYEE HAVING NAME = 'RONY'.}
\]

3.1.15. Updating an Entity or a Relationship
(ERROL 31)
\[
\text{UPDATE SUPPLY PRICE = 700 WHERE DEPARTMENT HAVING NAME = 'PRODUCTION', SUPPLIER STOCKING ITEM HAVING COLOR = 'RED'.}
\]

Remark: The last update will be executed for every item, since no restriction is specified for item. (The entity-set ITEM is participating in SUPPLY.)

3.1.16. Deletion of an Entity or a Relationship
(ERROL 32)
\[
\text{DELETE ITEM WHERE ITEM HAVING COLOR = 'RED'.}
\]

3.1.17. Inserting the query output into an entity or a relationship set.
"GET INTO" (entity-set-name|relationship-set-name) GET-CLAUSE ["."]
(ERROL 33)
\[
\text{GET INTO AUTO I# NAME COLOR OF ITEM HAVING TYPE = "AUTO"}
\]
4. THE SEMANTICS OF ERROL

The semantics of ERROL is defined by RRA [RA1]. 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 RAs' (see [ULL]). The main difference lies in an implicit join operation embedded in binary operations. This embedded join is due to references (see section 3.1). 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 AGGREGATE FUNCTION operators which can be viewed as both RRA's and RA's.

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

<table>
<thead>
<tr>
<th>Examples</th>
<th>ERROL (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>θ-join (parameters: =, /=, &gt;, &lt;, ≥, ≤)</td>
</tr>
<tr>
<td>...ITEM REQUESTED BY DEPARTMENT MANAGED BY EMPLOYEE...</td>
<td>Chaining</td>
<td>PRODUCT (reduces to Natural-join)</td>
</tr>
<tr>
<td>...DEPARTMENT TIS REQUESTING ITEM... AND MANAGED BY EMPLOYEE...</td>
<td>AND coordination</td>
<td>PRODUCT (reduces to intersection)</td>
</tr>
<tr>
<td>...DEPARTMENT TIS EMPLOYING EMPLOYEE... OR HAVING LOCALITY...</td>
<td>OR coordination</td>
<td>OR (Bordered union)</td>
</tr>
<tr>
<td>...DEPARTMENT NOT REQUESTING...</td>
<td>Negation</td>
<td>NOT (Complement)</td>
</tr>
<tr>
<td>...SET ITEM CONTAINS SET ITEM...</td>
<td>Set comparison, Universal quantifiers</td>
<td>Set-join (generalized division; parameters: =, /=, ≥, &lt;)</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 ERROL constructs but they are necessary: the first one to correctly construct RRA expressions (i.e., expressions that give the desired meaning of given ERROL constructs; the second to drop unnecessary attributes from relations taking part in an RRA expression.)
5. ERROL TRANSLATION

5.1. GENERAL DESCRIPTION

ERROL translation consists of two parts: in the first, ERROL is translated to RRA operations. In the second, the RRA operations are interpreted by C routine calls, or are executed as C programs. RRA (Reshaped Relational Algebra) is described in [MR2], [MR3], [RA1] in detail. RRA is equivalent to the regular relational algebra, but is more convenient as a target language for ERROL translation.

5.2. ERROL TO RRA TRANSLATION

The algorithm for translating ERROL to RRA defines an RRA operations sequence (an RRA expression) for every ERROL statement. The first step is to view the statement as a tree of syntactical elements. Then the syntactical elements are replaced by RRA operators which are the semantical elements. Traversing the tree in an appropriate order results in the desired RRA sequence of operators. The translation of ERROL is syntax directed.

5.2.1. REPRESENTATION OF AN ERROL STATEMENT AS A TREE

It is convenient to represent an ERROL statement by a "directed hyper-tree". This tree-hyper-graph (referred subsequently as "tree") will be defined in the next section. The tree direction is consistent with the direction of reading the query. It has one root and a directed path from the root to each of the leaves.

The translation algorithm to RRA consists of constructing the tree and then traversing the tree from leaves to root and right to left.

5.2.1.1. RULES FOR BUILDING THE TREE

Remark: In what follows, the directions of arcs are in the order of reading the ERROL sentence.

The tree is built according to the following rules while parsing the statement:

1. Every attribute or a name of an entity (the entity's identifier) is a node in the tree. The node is given a subscripted name \( X_i \) (\( i = 0, 1, 2, \ldots \)). The nodes are numbered consecutively according to the order of reaching them. If a name was referenced before, it does not get a new number, but the same number of the former occurrence. The new name and the entity or attribute name is associated to the node. Nodes referencing the same entity are given the same number. Constants are also nodes in the tree. The node representing a constant carries the constant's value.

2. All the nodes that belong to one basic statement are connected with a "wide" hyper-arc. Every hyper-arc carries an entity or relationship name, together with a subscripted name \( T_i \) (\( i = 0, 1, 2, \ldots \)). The wide arcs \( T_i \) are numbered in a similar way to the nodes \( X_i \). This "\( T_i \)" new name will replace the entity or relationship name. This hyper-arc shrinks to a simple arc if it represents a binary entity relationship. The hyper-arc is directed from the node representing the subject (input node) to the nodes representing the attributes (output nodes). subject (entity or relationship identifier) and the output nodes are the attributes.

(A) If the basic statement contains a role, then the relationship name must be given.
(B) If the basic statement signify owning, i.e. it contains the words HAVING or OF, then the entity name is given to the arc.

3. Every node is contained in at most one wide hyper-arc. Several nodes may be associated to an entity name (an entity identifier) appearing in the statement if this entity occurrence is connected with more than one role. These nodes are given the same subscripted name.

4. If the word SET is connected with an attribute name or an entity name, the appropriate node is a "fat" one.

5. If an aggregate function is connected with an attribute name or an entity name, then the appropriate node is a "fat" one (an implicit set). This node is connected to a node representing the result value of the aggregate function operation. The arc connecting the two nodes carries the aggregate function name.

6. A "narrow" hyper-arc (an arc which is not wide) contains two nodes, both of them regular or fat, or three regular nodes. It carries the following symbol:
   (A) A comparison symbol between two attributes or an attribute and a constant, having that symbol in the statement (these are regular nodes).
   (B) A set comparison symbol between two sets having elements of the same type for set comparison in the statement (fat nodes).
   (C) The symbol "+", if nodes represent the same entity (same $X_i$) appearing in two basic concatenated statements.
   (D) The symbol "not" if the word NOT appears in the statement. The input node is the result of the "not" operation and the output node is the node corresponding to the basic statement the negation is made upon.
   (E) The symbol "or" or "and" if there is an OR or an AND respectively in the statement. The two output nodes are those that belong to the basic statements with the union (or conjunction) between.

7. The last entity in the get-clause is the root of the tree. In this case, nodes may not represent a part of a basic statement (for example: ITEM in GET ITEM). Then the element (ITEM) is represented by a node ($X_i$) as usual, and a wide arc ($T_j$) (there is no additional node in the hyper-arc).

8. A name of a node which appears in the output is underlined.

9. A complex statement representing association between several attributes and a relationship set is represented by a forest.

Remark: Note that the tree corresponding to a statement without a branching is a tree without branches, (straight line).
5.2.1.2. REPRESENTING A STATEMENT BY A TREE - EXAMPLES

Example 1: Tree with no branches

GET SUPPLIER SUPPLING SET ITEM CONTAINS SET ITEM STOCKED BY SUPPLIER HAVING LOCALITY = "HAIFA"

Example 2: AND/OR branching on an entity

GET DEPARTMENT, ITEM TIS REQUESTED BY DEPARTMENT! OR SUPPLIED TO DEPARTMENT!

SUPPLIER:X₁ and SUPPLIER:X₄, and also ITEM:X₂ and ITEM:X₃ have different X₄ numbers because they reference different sets. Referencing the same set is expressed by the reference symbol (!). In such a case they will get the same X₄ number. In contradiction, SUPPLIER:X₄ has two nodes with the same name because it is a relationship set common to two
Example 3: AND/OR branching on sets
GET DEPARTMENT REQUESTING SET ITEM WHICH CONTAINS SET ITEM TIS HAVING COLOR = "RED";
AND CONTAINS SET ITEM STOCKED BY SUPPLIER HAVING SNO = 500

Example 4: Branching on non-binary relationships
GET DEPARTMENT REQUESTING ITEM SUPPLIED BY SUPPLIER HAVING LOCALITY = "PARIS" TO DEPARTMENT HAVING FLOOR = 2

Example 5: AND/OR branching on an attribute
GET DEPARTMENT SUPPLIED BY SUPPLIER HAVING NAME TIS = "YOSI" OR = "GIL"
5.2.2. DERIVING AN RRA OPERATIONS SEQUENCE FROM THE TREE

Representing an ERROL statement as a tree is the first step in its translation to an RRA (operations) sequence.

The second step is to produce out of the tree a sequence of RRA operations. Every arc in the tree represents an RRA operation. The translation is done by traversing the tree in DFS order.

The schemes of temporary relations and attributes tables (see also next section) can be computed together with the translation to an RRA sequence by a pass on the tree. The derived RRA sequence plus the schemes of all the temporary relations carry information equivalent to the information carried by the tree. This is the information needed to execute the algebraic expression. An alternative way to describe this information is to give the schemes of projections only, with counters associated to attributes and to the projections. The counters enable to compute the schemes of the other temporaries during the execution. In the following section the correlation between the tree elements and RRA operations is defined. Later, the algorithm to translate the tree to RRA is described.

5.2.2.1. CORRELATING TREE ELEMENTS WITH RRA OPERATIONS

Wide arc - is converted to a projection on necessary attributes only of a relation representing an entity or relationship set, together with renaming of the attributes (to $X_i$ names). Let $R$ be the entity or relationship set this arc represents, and $T_i$ the name of the arc. Then the operation is:

$$T_i = \text{Rename}(\text{Project}(R))$$

The fields this projection extracts are in the schemes table.

Narrow arc - is equivalent to an RRA operation other than projection-renaming. The operation type is decided according to the symbol associated with the arc. The operands of the operation are the temporary relations ($T_i$'s) that belong to the wide arcs this narrow arc connects, or a temporary and a constant when the narrow arc connects the wide arc with a node representing the constant.

The symbols are replaced by operations as follows:

1. **Comparison symbol** - when the arc with a comparison symbol connects nodes that represent attributes, the operation is a $\sigma$-join. Let $T_i, T_j$ be the temporaries this arc connects. Let $X_i, X_j$ be the attributes connected with the arc and $\sigma$ the comparison symbol. Then the RRA operation is:

$$T_k = T_i \Join_T_j \sigma X_i \theta X_j$$

When the arc connects an attribute and a constant, the operation is selection. Let $C$ be the constant and $X_i$ the attribute connected to $C$ with an arc. Let $T_i$ be the temporary $X_i$ belongs to. Then the operation is:

$$T_k = \sigma X_i \theta C(T_i)$$

2. **Symbol** - is equivalent to a Cartesian product between the temporaries this arc connects. Let $T_i, T_j$ be these temporaries. Then the operation is:

$$T_k = T_i \ast T_j$$

3. **Set comparison symbol** - is equivalent to a set-join operation. Let $T_i, T_j$ be the temporaries associated to the wide arcs connected by this arc. Let $X_i, X_j$ be the names of fat nodes that represent the sets, and let $\delta$ be the set comparison symbol. Then the operation is:
4. **Aggregate function name** - is equivalent to the appropriate aggregate function. Such an arc connects a fat node representing a set with a regular node representing a computed attribute which gets the value of the aggregate function computed on that set. Let \( T_i \) (a wide arc) be the temporary on which \( X_0 \) (a fat node) is grouped to sets. Then the RRA operation is:

\[
T_k = T_i \setminus \text{join} T_j
\]

\[
\left[ X_0 \right] \delta \left[ X_0 \right]
\]

5. **And/or/not** - Let \( T_i, T_j \) be the temporaries that were computed by the algebraic computation of the branches the logical operation is between.

- If the arc carries the word **AND**, a Cartesian product is made (which reduces automatically, to an intersection). This operation is:

\[
T_k = T_i \times T_j
\]

- If the arc carries the word **OR**, an or operation is executed:

\[
T_k = T_i \text{ or } T_j
\]

- Negation is defined on one branch. It is equivalent to the word **NOT** (the complement of \( T_i \) to the cartesian product of the entity sets in \( T_i \)):

\[
T_k = \text{not} \left\{ \left[ T_i \right] \right\}
\]

### 5.2.2.2. Evaluating Attribute and Scheme Tables

The evaluation can be done after the compilation was finished. It is done by passing the tree and the RRA operations computed.

#### 5.2.2.2.1. Evaluating the Attribute Table

1. Every attribute \( X_i \) has an entry in the table.

2. The original name (before renaming \( X_i \)) of the attribute will be the one its node carries. An aggregate function computed attribute name is composed of the function name and the grouped attribute name (e.g. the attribute "quantity" is summed up to become "sum_quantity").

3. The value of the attribute reference counter (C_counter) is equal to the number of appearances in the tree. If the attribute appears in the output, its counter is incremented by one. It is also incremented by one if the attribute is underlined a multiple number of times in the tree.

4. The value of operations reference counter (op_ref) is equal to the number of explicit appearances of the attribute in the RRA operations derived (the attribute appears as a parameter).

#### 5.2.2.2.2. Evaluating the Scheme Table

1. Every temporary \( T_i \) appearing in the RRA operations has an entry in the table.

2. The value of the temporary appearance counter (T_counter) is equal to the number of appearance of the temporary as an operand in the RRA sequence.

3. Every temporary belonging to a wide arc (i.e. projected temporaries) has its attributes names \( (X_i) \) predefined by the compiler. All the other temporaries will have their attributes decided at run time.
5.3. INTERPRETING RRA OPERATORS OR EMBEDDING THEM IN A C PROGRAM

In essence, the translation of RRA to C is very straightforward. Every RRA operation has its equivalent C routine that executes it, and the translation is done on a one-to-one basis.

For example:

```c
select(t2, t1, x1, "haifa", =)
```

in the output of the compiler is translated to the routine call

```c
select(2, 1, 1, "haifa", ");
```
6. SYSTEM ARCHITECTURE

6.1. INTRODUCTION

An end-user implementation of the language by the name of ERIS (Entity Relational Interface System) was developed in order to check the language ERROL and to test its compiler. It was implemented on top of the relational DBMS INGRES, using C, in VAX/UNIX environment. A description of ERIS appears in [ALP]. The current system, ES (Errol System), is a complete DBMS based on ERIS and built on top of SDBMS (Simple Data Base Management System - see [IMA], [NAV], [GOR]).

SDBMS is an improved file system based on B+ trees, that includes features of concurrency control and recovery. SDBMS as well as the entire ES were also developed in C (for documentation on C - see [KER]), VAX/UNIX environment (for documentation on UNIX - see [UNI]), with a special attention given to portability. An attempt was done to make the system be able to be passed freely between UNIX computer systems. As for UNIX-like systems like the PC-JX and VENIX on the IBM-XT, slight modifications are needed, especially in the full screen routines, and the file locking manager.

In order to implement ES on a micro computer, the size of basic elements of the software must be small enough. SDBMS stands well in this criterion. The kernel of SDBMS sizes about 57K. All the other parts together are approximately 150K. Thus, it can run on a micro with 320K memory or more (memory is allocated for the operating system and the data segment; also, a safety margin is assumed).

The system consists of five parts:

1. SDBMS.
2. ERROL to RRA compiler.
3. The RRA interpreter/compiler and the Entity-Relationship interface.
4. The user's monitor.
5. The full screen operations subsystem.

Addition of other parts, like form generator, report generator, C:ERROL programming tools, spreadsheet interface, are under design.

6.2. SDBMS - Simple Data Base Management System

6.2.1. GOALS

The SDBMS is actually a relational extended file management system oriented principally towards non expert users. Furthermore, one of its goals is to enable its operation without an expert database administrator.

To achieve this, simplicity was a keyword during the design of the system. The system maintenance and handling were made easy during its life cycle.

The system has features like: self documentation, simple algorithms and very uniform data structures which help all together to produce a simple, easy to use and understand system.

6.2.2. CONCURRENCY CONTROL AND RECOVERY

An application program is a set of operations that manipulate the database. These operations are organized as a sequence of one or more transactions. The transaction is the atomic consistent unit. A transaction begins with the begin_transaction command. Then follows the program that carries out the
required operations using the functions supplied for this purpose by the user interface. At the end comes the end_transaction command. It should be stressed that the transaction can contain any program of any size as the programmer wishes. A transaction is considered as done after a commit command is executed. It can also be aborted before being finished, either by the abort command or by system failure. Then it is undone (its effect on the database is canceled). Transactions cannot be nested.

The concurrency control mechanism is of the lock oriented type. It consists of two submechanisms: one for the index (the internal nodes of the B+ tree), and the other one for the data part (the leaves). The two submechanisms are independent, and the separation enables to increase drastically the degree of concurrency. The first submechanism is based on early splitting and merging of nodes in the B+ tree (see details in [MON]). The second submechanism is 2-PL (see [ULL]).

The granularity of the locks in the system is the physical block (page). This was made so mainly because of some recovery problems. Any smaller unit than a page involves a considerable more complex recovery system. Putting locks on this level, the log or audit trail can be built recording the actions performed at the physical level rather than at logical level. The before (or after) image record of the updated page is stored in the log avoiding in this form complex redo and undo routines at the logical level. It also helps keeping the lock table of the lock manager relatively small.

A transaction may be aborted by several reasons. The most obvious is a system crash in which all active transactions are interrupted. Other possible cause of abortion comes from the deadlock prevention system of the concurrency control. A time out mechanism causes the abortion of transactions that have been waiting too long to get a lock. Finally, a transaction can be aborted of logical reasons. Suppose, for instance, that the transaction was given incorrect input, data and cannot proceed with the updates.

The SDBMS recovery system covers soft faults. Faults involving losing of data stored on disk must be managed with the usual precaution measures of making backups regularly.

6.2.3. SDBMS PRIMITIVES

In order to access and manipulate the data base a set of primitive functions was defined. The applications programmer can access the database through this set of functions exclusively. The 4 commands mentioned in the previous section: begin_transaction, end_transaction, commit and abort are part of the SDBMS primitives.

The primitives available include:
- DB and relation opening.
- add, update and delete a tuple.
- keyed and sequential retrieval.
- relation definition.
- transaction primitives.

ES access the data base through the primitives to keep the connection between the SDBMS and ES as concentrated as possible. This primitives are used in spite of some loses in performance, in order to keep modularity of the system. However, we also access the open relation structure to retrieve information about the relation.
6.3. ERROL TO RRA COMPILER

The compiler consists of two main parts: the lexical analyser and the parser. The lexical analyser reads the statement and identifies the type of words appearing in the text. The lexical analyser associates to every word a token according to its type.

The parser gets the tokens sequence. In the sequence it identifies subsequences as syntactical elements and the roles the tokens play in it. The parser used is syntax-directed using an EBNF description of ERROL statements. According to the syntactical elements being identified, semantic subroutines are invoked in order to create the compiler outputs.

The outputs of the compiler are:
1. A sequence of RRA operations, describing an RRA expression.
2. A list of renamed attributes taking part in the expression, associated with references to the appropriate operands (relations) of the expression. For each of the above attributes two counters' values are given.
3. A list of schemes of the expression operands after projection and renaming (these are not all the temporary relations used). For each scheme the value of T_counter is given. This counter is used to determine when a temporary is obsolete at run-time.

6.4. RRA INTERPRETER/COMPILER

6.4.1. INTRODUCTION

The RRA interpreter takes the sequence of RRA operators generated by the ERROL compiler and interprets them as a sequence of subroutines calls. Each subroutine implements an RRA operation or data manipulation in terms of SDBMS primitives and utility functions (see list of these subroutines in Appendix D). Part of the data manipulation operations are done independently, and the others end an RRA sequence.

For example:

DESTROY SUPPLIER

is translated as destroy_ent("SUPPLIER");

A different example:
DELETE SUPPLIER WHERE SUPPLIER HAVING LOCALITY = "LONDON".

is translated as

project(1,"SUPPLIER");
select(2,1,"LONDON","=");
delete_ent("SUPPLIER",2);

The first two calls in the last sequence are the RRA operations project and select. The last one is a data manipulation command, deleting all tuples from entity SUPPLIER that reside in temporary T2.

We can also build (and store) a C program to execute the given query. The idea is very simple: the subroutines calls sequence described above is stored in a file together with appropriate prefix and suffix. The prefix includes the initial values of the two data structures CORRELATION and OP_SCHEMES (see next section). This gives us the query as a C program.

At the following sections we will describe the data structures, the data dictionary, the structure of the various kinds of relations, and the ideas standing behind the design of the relational algorithms.
6.4.2. RELATION STRUCTURE

There are two basic kinds of data relations: relations representing entity sets, and relationship sets. There are three more auxiliary types: dictionary relations, temporary relations, and field projections.

A description of the various types follows.

6.4.2.1. ENTITY RELATION

The entity relation - representing an entity set - is a regular relation (that is, it resides in the database and can be manipulated directly).

It has a key composed of several fields, all of them in the beginning of the tuple. The reason for this restriction is to make access easier and faster. It is not forced by any of the other parts of the system. The order of the key fields is determined by the order of key elements in the entity definition.

An entity set has a header line in DICTIONARY. The flag is 's'.

6.4.2.2. RELATIONSHIP RELATION

The relationship relation - representing a relationship set - is the second kind of a regular relation.

The key in a relationship is composed of the keys of the entities it connects. There may be non key fields as well.

A key of an entity is a concatenation of all the key fields in the entity, coded into a string. This manipulation is transparent to the user, who accesses and sees the entity key as divided into its components. The coding costs space, but saves lots of time and program length.

A relationship also has a header line. The flag is 'r'.

6.4.2.3. DATA DICTIONARY RELATIONS

There are 3 dictionary relations: RELATIONS, COLS and DICTIONARY. They do reside in the database but cannot be manipulated directly. Information can be retrieved from them as from entities, but they cannot be destroyed, for instance. They were already discussed.

6.4.2.4. TEMPORARY RELATIONS

Temporaries (temporary relations) play an important role in the relational operations. They are the input and the output of all the RRA operations (except project).

A temporary is created by a projection of a regular relation, or is the result of an RRA operation.

The temporary is placed in a regular file. The file is usually unordered (except after being sorted for some special usage) because an RRA operation accesses all tuples of the temporary (and the cheapest access of all the tuples is sequentially). The usage of a regular file is very efficient both in time and space. All the complex manipulations of building a B+ tree in the database for a temporary are avoided. There is also a considerable saving of time due to drastic load reduction in the concurrency control mechanism. There is no point to put temporaries in the database itself because the advantages of the database, such as recovery are inapplicable here. After a crash all the temporaries are cleared by one UNIX command ("rm"), and the query is executed from the start.

The file has a fixed record format: the fields' types are according to the types in DICTIONARY. It enables an in-core fast sort (i.e., quick-sort) if the file is
small enough to be sorted in memory. The temporaries have to be sorted in any RRA operation.

The information about the temporary scheme is placed in the table OP_SCHEMES. It contains an array of pointers into CORRELATION - that contains the description of each field.

One of the problems that has to be taken care of is concurrency control. There can be many temporaries with the same number (because there may be several users running on the same directory). A unique identifier must be given to each temporary. The process number of the monitor - always a unique identifier in UNIX - was chosen for this purpose. The name of temporary \( T_i \) in process \( j \) will be \( T_{ik} \) where

\[ k = 1000 \times j + i \]

6.4.2.5. FIELD PROJECTIONS

In some of the relational operations (or, not) a projection by some field is needed (actually, what is needed are domains of attributes). Because the projection by some field (correlation symbol) is the same in different temporaries, it is worthwhile to compute only one copy of it. This projection is computed in the first appearance of the field, in the operation "project". A flag in CORRELATION is kept to indicate if the field was projected.

The projection is kept in a file, in a format similar to that of a temporary. The problem of concurrency here is similar to that of the temporaries. The solution is also similar. The file name of field projection \( X_i \) in process \( j \) will be \( X_{ik} \) where

\[ k = 1000 \times j + i \]

6.4.3. RRA ALGORITHMS

No detailed description of the algorithms is given here, but rather the ideas behind. For a full description of the RRA algorithms see [IMA].

There are two kinds of RRA operations: project and the others. Project projects a regular relation into a temporary. It also projects the necessary fields. The other operations do not touch the database, and work solely with temporaries.

Project is a simple operation: we process the tuples sequentially one by one, and then drop non unique tuples from the result temporary.

The sort used is merge-sort. It has the best average time for sorting files. We used a modification of the UNIX sort utility.

Since the longest operations are based on sort, each of them takes at most \( O(n \log_2 n) \) time, where \( n \) is the maximal number of tuples in the input and output temporaries. Using sorted temporaries, every operation can be carried out sequentially.

Let us take for example "product". It is actually a natural join over all the identical fields. We first sort the two operand temporaries by the shared fields. Then we merge the two temporaries. Each pair of joinable tuples is put in the result file. The time of such an operation is linear in \( n \), so the total time is not more then \( O(n \log_2 n) \).

Because many of the RRA operations are independent, they can be carried out concurrently. First, all the project operations can be made concurrently. After that, every RRA operation that has its operands ready can be carried out immediately. It is done by forking a process for every new operation. This concurrent processing improves the overall performance.
6.5. USER’S MONITOR

The user’s monitor enables to activate the various system components interactively. The dialogue is carried out using a simple command language. Each command is prefixed with the symbol “/”, and is written at the monitor’s prompt. The ERROL statements are written in free format at the monitor’s prompt. The current monitor’s options reflect the fact that the system yet serves as a research tool, mainly for developing ERROL. Accordingly, the options enable the following:
1. Help options: specifying the scheme or detailed parts of it; displaying the manual.
2. Compilation and execution of ERROL statements, with/without compiler output or temporaries display; Storing of a C program.
3. Relation display.
4. ERROL statement editing.
5. Log control mechanism.
6. Direct editing of entities (the Visual option).
7. System recovery.

6.6. VISUAL EDITING OF AN ENTITY

One of the strongest properties of ERROL is the possibility to make mass changes in the data. Inserting a group of relationships obeying certain predicate can be done with a single statement. Mass delete is also possible.

There is an operation which is very clumsily done with ERROL. Every entity must be inserted in a separate statement. It will be much more convenient if we can insert tuples visually into an entity set. This was the motivation for the VISUAL option.

VISUAL is a full screen application that allows insertion, and update of entities. It also allows retrieval of tuples one by one by specifying a key value. VISUAL makes use of the screen packages Curses and Termllib (see [KEN]).

A key cannot be updated. The changes are done directly on the database and do not use the compiler or the interpreter. This results in a fast response time. Deletion is excluded because this command should obey the inclusion constraints governing the ERM, and hence is executed with an effect on tuples in other relations. Such command is left to be done by an ERROL statement.

To enter VISUAL, issue an appropriate command from the monitor environment (v e_set). VISUAL checks if the given parameter is really an entity name, and then displays a panel with all the columns on. Specified are the column name, column type, and the column value (or place for it).

The panel permits travel between the fields, insert, update and delete values as wished. Insert or update of the tuple on the screen is permitted. The program beeps if an error is detected. Retrieval of a tuple by key is also possible.

Most of the commands are control characters. Mnemonics are used as much as possible.

7. CONCLUSION

ES is an interactive system for managing a data base under the ERM using ERROL. ES was built in order to implement ERROL in a modular, user-friendly, efficient and compact way.
As the lower layer we used SDBMS, an improved file system based on $B^+$
trees, which includes features of concurrency control and recovery. The kernel
of the SDBMS is 57 KBytes; the other parts of ES are around 150K. Thus, it is
possible to transport the system to a personal computer (for example - IBM-XT).

The usage of UNIX files as temporary relations (instead of placing them in
the database) saves execution time. All the overhead of indexed retrieval and
update, and also lock managing is saved. The temporary relations (temporaries)
are processed sequentially each, so no indexing is needed. Processing each tem-
porary as a separate task enables concurrency within each transaction which,
in most cases, increases the overall performance. Every query has its own tem-
poraries, so no concurrency control or recovery mechanism is needed for them.
The problem of temporaries with the same index of different users was solved by
adding a unique user identifier (his main process id) to the index.

ES was built as modular as possible. The connections between its various
parts were kept as compact as possible.

Information retrieval from the data dictionary and from the actual data can
be done in a similar way. This makes the system easier to use.

The user's monitor includes options for editing, compiling and executing an
ERROL transaction. Besides, it has many more options, such as compiling (and
storing) a transaction, and direct data entry (entity editor), which overtake the
ERROL entity insert.

RRA algorithms were developed from Relational Algebra algorithms. The
execution time of every RRA algorithm is not greater than $O(n \log n)$, where $n$
is the number of tuples in the largest relation taking part in the operation (as
input or output).

The user key is also the physical key. This option was chosen in order to
take advantage of the fast access by key mechanism provided by the $B^+$
tree, and of the sorted sequential order induced by the $B^+$ tree. A coded version
of the entity key is used in the relationships.

ES was built as a tool for further development of EROIL and as a base for an
applications generator using EROIL as its main manipulation language.
REFERENCES


APPENDIX A

ES USER'S MANUAL

To create a database with the arbitrary name "demo" type the command:

`% newdb demo`

This requests the "UNIX" to invoke a special procedure which creates an ES database called "demo" using SDBMS. If the database already exists, the user will get the message:

`Database demo already exists`

Otherwise he will get the message:

`Database demo was created!`

and will become the DBA of the database "demo".

To destroy a database type the command:

`% rm demo`

To start using ES type the command:

`% es demo`

This requests the "UNIX" to invoke ES using the database called "demo". After a few seconds the following may appear:

Sun Feb 12 12:15:25 1984

Welcome to the Entity_Relationship database: demo

Type 'o' for options.

> .sp 2

The ES monitor prints the sign ">" at the beginning of each line to remind you that ES is waiting for input.

Two types of input to the system: ES commands and ERROL sentences. ES commands are prefixed with "/". ERROL sentences are given in a free format. Each command will be described together with Examples of usage if possible.

1. o

This command displays the options available to the user (including options of ES commands and of ERROL sentences).

2. u

This command prints out the Reference Manual on the user's terminal.

3. s[+s#]
This command consists of two options:

I. s

Prints the ERROL sentence placed in a file called "Data". This sentence is called the "Current Sentence", and is the sentence which the user can compile, execute or edit.

II. s+ s#

All the sentences inserted by the user are saved in a file called "history". This command prints the number s# sentence from the file. Example:

> /s+4 (Print the 4th sentence.)

4. l

This command prints the contents of the file "history" which contains a list of sentences inserted by the user, since the last deletion of the file (see 'd').

5. d

This command deletes the contents of the file "history". All the sentences except the current sentence are deleted. The current sentence is placed at the beginning of the file.

6. m s#

This command moves the number s# sentence, in the file "history", to the file "data". This sentence will be the current sentence. Example:

> /m2 (Move the 2nd sentence to become the Current Sentence).

7. c[+]

This command calls the "ERROL to RRA" compiler. The compiler takes the current sentence which is placed in the file "Data" as an input. Compilation errors will be printed on the terminal. If the compilation terminates without errors, the compiler gives a sequence of RRA (Reshaped Relational Algebra) operators as an output.

The user may have the compiler output printed on his terminal. This is done when the ' + ' option is used.

8. x[+]

This command executes the sequence of RRA operations produced by the compiler. The user may use this command after compiling the Current Sentence. After execution, the result relation is printed on the terminal.

The user may have all temporary results printed on his terminal but using the ' + ' option.

9. r[+]

This command compiles and executes the current sentence. This is equivalent to executing the command 'c' followed by 'x[+]'.

10. b b#

This command builds a C file in the form of "Qxxxxx.cc", where xxxxx is the process number * 1000 + b#, that contains the source of a C program which is a
translation of the current ERROL statement. The file should be compiled with libraries E.LIB and E_LIB.

11. e

An editor for the current sentence: This command calls the UNIX Editor "Vi" for the file "Data" in which the current sentence is placed. To quit the editor type ':wq'.

12. h[+ {e_set|r_set}]

Help: The Entity_Relationship scheme of the Data_base "demo". This command consists of two options:

I. h

Tells the system to print on the terminal the names of all the Entity_sets and of all the Relationship_sets existing in the Data_base.

II. h+ rname

Prints the scheme of an Entity_set or a Relationship_set relation whose name is 'rname'. In case of an Entity_set, the system will print a list of all the attributes that belong to the Entity_set together with the format of each attribute. Then another list of all the Relationship_sets in which this Entity_set is participating, together with the role played by this Entity_set in each Relationship_set. In case of a Relationship_set, the system will print a list of all the Entity_sets participating in this Relationship_set, together with the role played by each Entity_set. Then a list of the attributes belongs to the Relationship_set together with the format of each attribute. Example:

> h+ item (Print the scheme of the Entity_set 'item').

13. p {e_set|r_set}

Prints the contents of an Entity_set_relation or a relationship_set_relation (including temporary relationship relation) existing in the Data_base "demo". Example:

> p item (Print the contents of the Entity_set_relation 'item').

14. y

Recovery from SDBMS ERROR, or system crash. The ES system is implemented on the top of the Data_base management system SDBMS.

When this happens a recovery procedure has to be executed to prevent other SDBMS ERRORs derived From the first error.

This command executes the recovery procedure.

15. q

Quit ES.

16. v e_set

This command enter Visual mode. It starts editing the given e_set. You can not edit a relationship set. In this mode you can retrieve, add and update tuples. The reason for this addition is because ERROL does not support fast insertion of
entities. A list of Visual mode command follows:

LIST OF VISUAL OPTIONS

Control P: Redraw the screen.

Tab: Jump to the next field.

Control A:
  Toggle the insert state. Typing characters in insert mode shifts the other characters to the right. Otherwise they are overwritten.

Control H:
  Move a character backward (actually backspace).

Control I:
  Move a character forward.

Control K:
  Move a character upward.

Control J:
  Move a character downward.

Control D:
  Delete the character under the cursor.

Escape:
  Exit from VISUAL.

Control R:
  Clear the tuple.

Control G:
  Try to retrieve a tuple according to the information in the key.

Control U:
  Try to update the tuple. If it fails, it beeps and points to the first problematic field.

Control F:
  The same as Control U, only try to insert.
APPENDIX B

SAMPLE SESSION

Script started on Wed May 28 18:42:35 1986
Wed May 28 18:42:43 JST 1986
Welcome to the entity_relationship database: oer
Type `/o` for options.
> /y
Executing:

restart process complete. Actives and Audit files were erased
> /h

transaction begins, id number is : 1

ENTITY_SETS:
------------------
supplier
employee
department
item

RELATIONSHIP_SETS:
-------------------
stock
supply
employment
management
request

transaction 1 completes
> /h+item

transaction begins, id number is : 2

item entity_set:
================

Having attributes:
-------------------
ino --- c5 KEY
name --- c10
color --- c10
type --- c1

Participating in relationship_sets:

stock
supply
request

transaction 2 completes

> /p item

transaction begins, id number is: 3

item entity

<table>
<thead>
<tr>
<th>ino</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>A</td>
</tr>
<tr>
<td>12347</td>
<td>screw</td>
<td>green</td>
<td>B</td>
</tr>
</tbody>
</table>

transaction 3 completes

> /p supplier

transaction begins, id number is: 4

supplier entity

<table>
<thead>
<tr>
<th>sno</th>
<th>name</th>
<th>locality</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111</td>
<td>omer</td>
<td>haifa</td>
</tr>
<tr>
<td>22222</td>
<td>yosi</td>
<td>haifa</td>
</tr>
<tr>
<td>33333</td>
<td>moshe</td>
<td>ako</td>
</tr>
</tbody>
</table>

transaction 4 completes

> /p stock

transaction begins, id number is: 5

stock relationship

<table>
<thead>
<tr>
<th>sno</th>
<th>ino</th>
<th>quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111</td>
<td>12345</td>
<td>1</td>
</tr>
<tr>
<td>11111</td>
<td>12346</td>
<td>1</td>
</tr>
<tr>
<td>11111</td>
<td>12347</td>
<td>1</td>
</tr>
</tbody>
</table>
transaction 5 completes

> get supplier stocking set item contains set item
> /s

********** CURRENT SENTENCE **********

# 19

get supplier stocking set item contains set item

> /c+

transaction begins, id number is: 6

transaction 6 completes

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

The sequence of operators:

project(t2 ,item)
project(t1 ,stock)
divide(t3 ,t1 ,t2 ,x1 ,x2 ,contains)
print_in(t3 ,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>supplier</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>x1</td>
<td>item</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>x2</td>
<td>item</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

The operational scheme:

relid  counter  c_symbols
-------  --------  --------
t1      1         x0 , x1
t2      1         x2

> /x4

transaction begins, id number is: 7

item entity

<table>
<thead>
<tr>
<th>lino</th>
<th>name</th>
<th>color</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
transaction 7 completes

get supplier tis having name="ofe" and
stocking item having color="red"

transaction begins, id number is: 8
transaction 8 completes

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

The sequence of operators:
----------------------------------------------
project(t2,supplier)
select(t3,t2,x1,"ofe",:)=
project(t5,item)
select(t6,t5,x3,"red",:)=
project(t4,stock)
product(t7,t6,t4)
product(t8,t3,t7)
print_in(t8,nil)

The correlation symbols:
----------------------------------------------
c_symbol   org_name   c_counter op_ref
---  ---------      -----  ----
  x0 supplier     3       0
  x1 name         1       1
  x2 item         2       0
  x3 color        1       1

The operational scheme:
----------------------------------------------
re_id   counter   c_symbols
---  ---  ---
t2     1     x0  x1
tr     1     x0  x2
tr     1     x2  x3

> /+x+

transaction begins, id number is: 9

supplier entity

<table>
<thead>
<tr>
<th>sno</th>
<th>name</th>
<th>locality</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11111</td>
<td>ofer</td>
<td>haifa</td>
</tr>
<tr>
<td>22222</td>
<td>yosi</td>
<td>haifa</td>
</tr>
<tr>
<td>33333</td>
<td>moshe</td>
<td>akd</td>
</tr>
</tbody>
</table>
### t2 relation

<table>
<thead>
<tr>
<th>(x_0)</th>
<th>(x_1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>offer</td>
<td>yosi</td>
</tr>
<tr>
<td>yosi</td>
<td>moshe</td>
</tr>
</tbody>
</table>

### t3 relation

<table>
<thead>
<tr>
<th>(x_0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111</td>
</tr>
</tbody>
</table>

### item entity

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

### t5 relation

<table>
<thead>
<tr>
<th>(x_2)</th>
<th>(x_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345</td>
<td>red</td>
</tr>
<tr>
<td>12346</td>
<td>blue</td>
</tr>
<tr>
<td>12347</td>
<td>green</td>
</tr>
</tbody>
</table>

### t6 relation

<table>
<thead>
<tr>
<th>(x_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345</td>
</tr>
</tbody>
</table>

### stock relationship

<table>
<thead>
<tr>
<th>sno</th>
<th>ino</th>
<th>quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111</td>
<td>12345</td>
<td>1</td>
</tr>
<tr>
<td>11111</td>
<td>12346</td>
<td>1</td>
</tr>
<tr>
<td>11111</td>
<td>12347</td>
<td>1</td>
</tr>
<tr>
<td>22222</td>
<td>12345</td>
<td>1</td>
</tr>
<tr>
<td>22222</td>
<td>12346</td>
<td>1</td>
</tr>
<tr>
<td>22222</td>
<td>12347</td>
<td>1</td>
</tr>
</tbody>
</table>
t4 relation

<table>
<thead>
<tr>
<th>x0</th>
<th>x2</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111</td>
<td>12345</td>
</tr>
<tr>
<td>11111</td>
<td>12346</td>
</tr>
<tr>
<td>11111</td>
<td>12347</td>
</tr>
<tr>
<td>22222</td>
<td>12345</td>
</tr>
<tr>
<td>22222</td>
<td>12346</td>
</tr>
<tr>
<td>22222</td>
<td>12347</td>
</tr>
</tbody>
</table>

transaction 9 completes

> get supplier tis having locality != locality
> of supplier having name = "ofer"
> /c+

transaction begins, id number is : 10
transaction 10 completes

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

The sequence of operators : 
----------------------------------
project(t4 , supplier)
select(t5 , t4 , x4 , "ofer" , =)
project(t3 , supplier)
product(t6 , t5 , t3)
project(t2 , supplier)
t_join(t7 , t2 , t6 , x1 , x2 , ! =)
print_in(t7 , 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>supplier</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>x1</td>
<td>locality</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>x2</td>
<td>locality</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>x3</td>
<td>supplier</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>x4</td>
<td>name</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The operational scheme:

<table>
<thead>
<tr>
<th>relid</th>
<th>counter</th>
<th>c_symbols</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>
</tbody>
</table>

> /x+

transaction begins, id number is: 1
supplier entity

<table>
<thead>
<tr>
<th>sno</th>
<th>name</th>
<th>locality</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111</td>
<td>ofer</td>
<td>haifa</td>
</tr>
<tr>
<td>22222</td>
<td>yosi</td>
<td>haifa</td>
</tr>
<tr>
<td>33333</td>
<td>moshe</td>
<td>akko</td>
</tr>
</tbody>
</table>

t4 relation

<table>
<thead>
<tr>
<th>x3</th>
<th>x4</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111</td>
<td>ofer</td>
</tr>
<tr>
<td>22222</td>
<td>yosi</td>
</tr>
<tr>
<td>33333</td>
<td>moshe</td>
</tr>
</tbody>
</table>

t5 relation

<table>
<thead>
<tr>
<th>x3</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111</td>
</tr>
</tbody>
</table>

supplier entity

<table>
<thead>
<tr>
<th>sno</th>
<th>name</th>
<th>locality</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sno</td>
<td>name</td>
<td>locality</td>
</tr>
<tr>
<td>-----</td>
<td>-------</td>
<td>----------</td>
</tr>
<tr>
<td>1111</td>
<td>ofer</td>
<td>haifa</td>
</tr>
<tr>
<td>2222</td>
<td>yosi</td>
<td>haifa</td>
</tr>
<tr>
<td>3333</td>
<td>moshe</td>
<td>ako</td>
</tr>
</tbody>
</table>

**t3 relation**

<table>
<thead>
<tr>
<th>x2</th>
<th>x3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ako</td>
<td>3333</td>
</tr>
<tr>
<td>haifa</td>
<td>1111</td>
</tr>
<tr>
<td>haifa</td>
<td>2222</td>
</tr>
</tbody>
</table>

**t6 relation**

<table>
<thead>
<tr>
<th>x2</th>
</tr>
</thead>
<tbody>
<tr>
<td>haifa</td>
</tr>
</tbody>
</table>

**supplier entity**

<table>
<thead>
<tr>
<th>sno</th>
<th>name</th>
<th>locality</th>
</tr>
</thead>
<tbody>
<tr>
<td>----</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1111</td>
<td>offer</td>
<td>haifa</td>
</tr>
<tr>
<td>2222</td>
<td>yosi</td>
<td>haifa</td>
</tr>
<tr>
<td>3333</td>
<td>moshe</td>
<td>ako</td>
</tr>
</tbody>
</table>

**t2 relation**

<table>
<thead>
<tr>
<th>x0</th>
<th>x1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>haifa</td>
</tr>
<tr>
<td>2222</td>
<td>haifa</td>
</tr>
<tr>
<td>3333</td>
<td>ako</td>
</tr>
</tbody>
</table>

**t7 relation**

<table>
<thead>
<tr>
<th>x0</th>
</tr>
</thead>
<tbody>
<tr>
<td>33333</td>
</tr>
</tbody>
</table>

**t7 relation**

<table>
<thead>
<tr>
<th>supplier</th>
</tr>
</thead>
<tbody>
<tr>
<td>33333</td>
</tr>
</tbody>
</table>
transaction 11 completes

 transaction begins, id number is: 12
 transaction 12 completes

 The sequence of operators:

 \[ \text{project}\left(\text{t1, item}\right) \]
 \[ \text{count}\left(\text{t2, t1, x1, x0}\right) \]
 \[ \text{print in}\left(\text{t2, nil}\right) \]

 The correlation symbols:

 \[ \begin{array}{cccc}
 \text{c\_symbol} & \text{org\_name} & \text{c\_counter} & \text{op\_ref} \\
 \hline
 x0 & \text{item} & 2 & 1 \\
 x1 & \text{cnt\_item} & 2 & 0 \\
 \end{array} \]

 The operational scheme:

 transaction begins, id number is: 13

 item entity

<table>
<thead>
<tr>
<th>\text{ino}</th>
<th>\text{name}</th>
<th>\text{color}</th>
<th>\text{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>A</td>
</tr>
<tr>
<td>12347</td>
<td>screw</td>
<td>green</td>
<td>B</td>
</tr>
</tbody>
</table>

 t1 relation

 | \text{x0} |
 | 12345 |
 | 12346 |
 | 12347 |

 t2 relation

 | \text{x1} |
transaction 13 completes

\[ \text{get supplier!x stocking item supplied by supplier!x} \]

transaction begins, id number is: 14

transaction 14 completes

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

The sequence of operators:

- \text{project(t2, stock)}
- \text{project(t1, stock)}
- \text{product(t3, t2, t1)}
- \text{print_in(t3, nil)}

The correlation symbols:

\begin{tabular}{llll}
\text{c_symbol} & \text{org_name} & \text{c_counter} & \text{op_ref} \\
\hline
x0 & supplier & 3 & 0 \\
x1 & item & 2 & 0 \\
\end{tabular}

The operational scheme:

\begin{tabular}{llll}
\text{relid} & \text{counter} & \text{c_symbols} \\
\hline
t1 & 1 & x0 x1 \\
t2 & 1 & x1 x0 \\
\end{tabular}

transaction begins, id number is: 15

supply relationship

\begin{tabular}{llllll}
\text{sno} & \text{line} & \text{dno} & \text{quantity} & \text{price} \\
\hline
\end{tabular}
t2 relation

<table>
<thead>
<tr>
<th>x1</th>
<th>x0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

stock relationship

<table>
<thead>
<tr>
<th>sno</th>
<th>line</th>
<th>quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11111</td>
<td>12345</td>
<td>1</td>
</tr>
<tr>
<td>11111</td>
<td>12346</td>
<td>1</td>
</tr>
<tr>
<td>11111</td>
<td>12347</td>
<td>1</td>
</tr>
<tr>
<td>22222</td>
<td>12345</td>
<td>1</td>
</tr>
<tr>
<td>22222</td>
<td>12346</td>
<td>1</td>
</tr>
<tr>
<td>22222</td>
<td>12347</td>
<td>1</td>
</tr>
</tbody>
</table>

t1 relation

<table>
<thead>
<tr>
<th>x0</th>
<th>x1</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111</td>
<td>12345</td>
</tr>
<tr>
<td>11111</td>
<td>12346</td>
</tr>
<tr>
<td>11111</td>
<td>12347</td>
</tr>
<tr>
<td>22222</td>
<td>12345</td>
</tr>
<tr>
<td>22222</td>
<td>12346</td>
</tr>
<tr>
<td>22222</td>
<td>12347</td>
</tr>
</tbody>
</table>

trans3 relation

<table>
<thead>
<tr>
<th>x0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

trans3 relation

<table>
<thead>
<tr>
<th>supplier</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

transaction 15 completes

> get item tis not stocked by supplier
> /c+

transaction begins, id number is : 16
transaction 16 completes

*******************************************************************

The sequence of operators :
project(t2, stock)
not(t3, t2)
print_in(t3, 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>item</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

The operational scheme:

<table>
<thead>
<tr>
<th>relid</th>
<th>counter</th>
<th>c_symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td>t2</td>
<td>1</td>
<td>x0</td>
</tr>
</tbody>
</table>

transaction begins, id number is: 17

stock relationship

<table>
<thead>
<tr>
<th>sno</th>
<th>lno</th>
<th>quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1111</td>
<td>12345</td>
<td>1</td>
</tr>
<tr>
<td>1111</td>
<td>12346</td>
<td>1</td>
</tr>
<tr>
<td>1111</td>
<td>12347</td>
<td>1</td>
</tr>
<tr>
<td>2222</td>
<td>12345</td>
<td>1</td>
</tr>
<tr>
<td>2222</td>
<td>12346</td>
<td>1</td>
</tr>
<tr>
<td>2222</td>
<td>12347</td>
<td>1</td>
</tr>
</tbody>
</table>

t2 relation

<table>
<thead>
<tr>
<th>x0</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345</td>
</tr>
<tr>
<td>12346</td>
</tr>
<tr>
<td>12347</td>
</tr>
</tbody>
</table>

t3 relation

<table>
<thead>
<tr>
<th>x0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>item</th>
</tr>
</thead>
</table>
transaction 17 completes

transaction begins, id number is: 18

transaction 18 completes

get supplier stocking set item which
contains set item having color = "red"
and contains set item stocked by supplier tis
(having name = "of er"
or having locality tis = "haifa" or = "ako")
and not supplying department having name =
"engineering";
or eq set item having type = "A"

transaction begins, id number is: 19
transaction 19 completes

*************** COMPILER OUTPUT ***************
The sequence of operators:
-----------------------------------------
project(t2, item)
select(t3, t2, x2, "red", =)
project(t1, stock)
divide(t4, t1, t3, x1, x2, contains)
project(t6, supplier)
select(t7, t6, x6, "of er", =)
project(t8, supplier)
select(t9, t8, x7, "haifa", =)
select(t10, t8, x7, "ako", =)
or(t11, t9, t10)
or(t12, t7, t11)
project(t14, department)
select(t15, t14, x9, "engineering", =)
project(t13, supply)
product(t16, t15, t13)
not(t17, t16)
product(t18, t12, t17)
project(t5, stock)
product(t19, t18, t5)
divide(t20, t1, t19, x1, x4, contains)
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>supplier</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>x1</td>
<td>item</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>x2</td>
<td>item</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>x3</td>
<td>color</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>x4</td>
<td>item</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>x5</td>
<td>supplier</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>x6</td>
<td>name</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>x7</td>
<td>locality</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>x8</td>
<td>department</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>x9</td>
<td>name</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>x10</td>
<td>item</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>x11</td>
<td>type</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

The operational scheme:

<table>
<thead>
<tr>
<th>relid</th>
<th>counter</th>
<th>c_symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>3</td>
<td>x0 x1</td>
</tr>
<tr>
<td>t2</td>
<td>1</td>
<td>x2 x3</td>
</tr>
<tr>
<td>t5</td>
<td>1</td>
<td>x4 x5</td>
</tr>
<tr>
<td>t6</td>
<td>1</td>
<td>x5 x6</td>
</tr>
<tr>
<td>t8</td>
<td>2</td>
<td>x5 x7</td>
</tr>
<tr>
<td>t13</td>
<td>1</td>
<td>x5 x8</td>
</tr>
<tr>
<td>t14</td>
<td>1</td>
<td>x8 x9</td>
</tr>
<tr>
<td>t22</td>
<td>1</td>
<td>x10 x11</td>
</tr>
</tbody>
</table>

transaction begins, id number is: 20

> /x

t25 relation

<table>
<thead>
<tr>
<th>supplier</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111</td>
</tr>
<tr>
<td>22222</td>
</tr>
</tbody>
</table>

transaction 20 completes

> insert stock quantity=2 where supplier having sno="33333",
> item having type = "A"
transaction begins, id number is 21
transaction 21 completes

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

The sequence of operators:

project(t1, supplier)
select(t2, t1, x1, "33333", =)
project(t3, item)
select(t4, t3, x3, "A", =)
product(t5, t2, t4)
insert_rel(stock : t5 : quantity=2)

The correlation symbols:

<table>
<thead>
<tr>
<th>_symbol</th>
<th>org_name</th>
<th>_counter</th>
<th>op_ref</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>supplier</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>x1</td>
<td>sno</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>x2</td>
<td>item</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>x3</td>
<td>type</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The operational scheme:

relid  | counter | _symbols
-------|---------|---------
  t1    | 1       | x0 x1  
  t3    | 1       | x2 x3  

transaction begins, id number is 23
supplier entity

<table>
<thead>
<tr>
<th>sno</th>
<th>name</th>
<th>locality</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111</td>
<td>ofer</td>
<td>haifa</td>
</tr>
<tr>
<td>22222</td>
<td>yosi</td>
<td>haifa</td>
</tr>
<tr>
<td>33333</td>
<td>moshe</td>
<td>ako</td>
</tr>
</tbody>
</table>

t1 relation

<table>
<thead>
<tr>
<th>x0</th>
<th>x1</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111</td>
<td>11111</td>
</tr>
<tr>
<td>22222</td>
<td>22222</td>
</tr>
<tr>
<td>33333</td>
<td>33333</td>
</tr>
</tbody>
</table>
$t_2$ relation

\[
\begin{array}{c|c|c|c|}
| x_0 | & | | & | | \\
| 12345 | & | | | \\
| 12346 | & | | | \\
| 12347 | & | | | \\
\end{array}
\]

Item entity

\[
\begin{array}{c|c|c|c|}
| ino | name | color | type | \\
| 12345 | auto | red | A | \\
| 12346 | bolt | blue | A | \\
| 12347 | screw | green | B | \\
\end{array}
\]

$t_3$ relation

\[
\begin{array}{c|c|}
| x_2 | x_3 | \\
| 12345 | A | \\
| 12346 | A | \\
| 12347 | B | \\
\end{array}
\]

$t_4$ relation

\[
\begin{array}{c|}
| x_2 | \\
| 12345 | \\
| 12346 | \\
\end{array}
\]

$t_5$ relation

\[
\begin{array}{c|c|}
| x_0 | x_2 | \\
| 33333 | 12345 | \\
| 33333 | 12346 | \\
\end{array}
\]

Stock relationship

\[
\begin{array}{c|c|c|}
| sno | ino | quantity | \\
| 11111 | 12345 | 1 | \\
| 11111 | 12346 | 1 | \\
| 11111 | 12347 | 1 | \\
| 22222 | 12345 | 1 | \\
| 22222 | 12346 | 1 | \\
\end{array}
\]
transaction 22 completes
> insert supplier sno="A",name="ofer",locality="haifa"
> /c+

transaction begins, id number is: 23
transaction 23 completes

*************** COMPILÉR OUTPUT ***************
The sequence of operators:
-----------------------------
insert_ent(supplier :sno="A",name="ofer",locality="haifa")
> /x+

transaction begins, id number is: 24
supplier entity

<table>
<thead>
<tr>
<th>sno</th>
<th>name</th>
<th>locality</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>refer</td>
<td>haifa</td>
</tr>
<tr>
<td>11111</td>
<td>refer</td>
<td>haifa</td>
</tr>
<tr>
<td>22222</td>
<td>yosi</td>
<td>haifa</td>
</tr>
<tr>
<td>33333</td>
<td>moshe</td>
<td>ako</td>
</tr>
<tr>
<td></td>
<td>refer</td>
<td>haifa</td>
</tr>
</tbody>
</table>

transaction 24 completes
> update supplier locality="ako" where supplier having sno="A"
> /c+

transaction begins, id number is: 25
transaction 25 completes

*************** COMPILÉR OUTPUT ***************
The sequence of operators:
-----------------------------
project(t1 ,supplier)
select(t2 ,t1 ,x1 ,"A" ,=)
update_ent(supplier :t2 :locality="ako")

The correlation symbols:
The operational scheme:

<table>
<thead>
<tr>
<th>relid</th>
<th>counter</th>
<th>c_symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>1</td>
<td>x0  \ x1</td>
</tr>
</tbody>
</table>

transaction begins, id number is: 26

supplier entity

<table>
<thead>
<tr>
<th>sno</th>
<th>name</th>
<th>locality</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111</td>
<td>ofer</td>
<td>haifa</td>
</tr>
<tr>
<td>22222</td>
<td>yosi</td>
<td>haifa</td>
</tr>
<tr>
<td>33333</td>
<td>moshe</td>
<td>ako</td>
</tr>
<tr>
<td>A</td>
<td>offer</td>
<td>haifa</td>
</tr>
</tbody>
</table>

relation t1

<table>
<thead>
<tr>
<th>x0</th>
<th>x1</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111</td>
<td>11111</td>
</tr>
<tr>
<td>22222</td>
<td>22222</td>
</tr>
<tr>
<td>33333</td>
<td>33333</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
</tr>
</tbody>
</table>

relation t2

<table>
<thead>
<tr>
<th>x0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
</tbody>
</table>

supplier entity

<table>
<thead>
<tr>
<th>sno</th>
<th>name</th>
<th>locality</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111</td>
<td>ofer</td>
<td>haifa</td>
</tr>
<tr>
<td>22222</td>
<td>yosi</td>
<td>haifa</td>
</tr>
<tr>
<td>33333</td>
<td>moshe</td>
<td>ako</td>
</tr>
<tr>
<td>A</td>
<td>offer</td>
<td>ako</td>
</tr>
</tbody>
</table>
transaction 26 completes
> delete supplier where supplier having sno="A"
> /c+

transaction begins, id number is : 27
transaction 27 completes

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

The sequence of operators:
--------------------------
project(t1 , supplier)
select(t2 , t1 , x1 , "A" , =)
delete_ent(supplier : t2 )

The correlation symbols:
----------------------
c_symbol org_name c_counter op_ref
---------- --------- ------- -------
x0 supplier 2 0
x1 sno 1 1

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

transaction begins, id number is : 28

supplier entity

<table>
<thead>
<tr>
<th>sno</th>
<th>name</th>
<th>locality</th>
</tr>
</thead>
<tbody>
<tr>
<td>____</td>
<td>______</td>
<td>_______</td>
</tr>
<tr>
<td>11111</td>
<td>omer</td>
<td>haifa</td>
</tr>
<tr>
<td>22222</td>
<td>yosi</td>
<td>haifa</td>
</tr>
<tr>
<td>33333</td>
<td>moshe</td>
<td>akko</td>
</tr>
<tr>
<td>A</td>
<td>omer</td>
<td>akko</td>
</tr>
<tr>
<td>____</td>
<td>______</td>
<td>_______</td>
</tr>
</tbody>
</table>

t1 relation

<table>
<thead>
<tr>
<th>x0</th>
<th>x1</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111</td>
<td>11111</td>
</tr>
<tr>
<td>22222</td>
<td>22222</td>
</tr>
<tr>
<td>33333</td>
<td>33333</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
</tr>
</tbody>
</table>
t2 relation

| x0 | A |

supplier entity

<table>
<thead>
<tr>
<th>sno</th>
<th>name</th>
<th>locality</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>offer</td>
<td>haifa</td>
</tr>
<tr>
<td>1111</td>
<td>yosi</td>
<td>haifa</td>
</tr>
<tr>
<td>2222</td>
<td>moshe</td>
<td>ako</td>
</tr>
</tbody>
</table>

transaction 28 completes

> define entity auto key ino=c5,color=c10,type=c1
> /r

transaction begins, id number is: 29
transaction 29 completes

> get into auto ino color type of item having name="auto"
> /c+

transaction begins, id number is: 30
transaction 30 completes

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

The sequence of operators:

project(t2,item)
select(t3,t2,x4,"auto",=)
project(t1,item)
product(t4,t3,t1)
print_in(t4,auto)

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>ino</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>x1</td>
<td>color</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>x2</td>
<td>type</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>x3</td>
<td>item</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>x4</td>
<td>name</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The operational scheme:
transaction begins, id number is: 31

item entity

<table>
<thead>
<tr>
<th>ino</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>A</td>
</tr>
<tr>
<td>12347</td>
<td>screw</td>
<td>green</td>
<td>B</td>
</tr>
</tbody>
</table>

t2 relation

<table>
<thead>
<tr>
<th>x3</th>
<th>x4</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345</td>
<td>auto</td>
</tr>
<tr>
<td>12346</td>
<td>bolt</td>
</tr>
<tr>
<td>12347</td>
<td>screw</td>
</tr>
</tbody>
</table>

t3 relation

<table>
<thead>
<tr>
<th>x3</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345</td>
</tr>
</tbody>
</table>

item entity

<table>
<thead>
<tr>
<th>ino</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>A</td>
</tr>
<tr>
<td>12347</td>
<td>screw</td>
<td>green</td>
<td>B</td>
</tr>
</tbody>
</table>

t1 relation

<table>
<thead>
<tr>
<th>x0</th>
<th>x1</th>
<th>x2</th>
<th>x3</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345</td>
<td>red</td>
<td>A</td>
<td>12345</td>
</tr>
<tr>
<td>12346</td>
<td>blue</td>
<td>A</td>
<td>12346</td>
</tr>
<tr>
<td>12347</td>
<td>green</td>
<td>B</td>
<td>12347</td>
</tr>
</tbody>
</table>
t4 relation
<table>
<thead>
<tr>
<th>x0</th>
<th>x1</th>
<th>x2</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345</td>
<td>red</td>
<td>A</td>
</tr>
</tbody>
</table>

auto entity

<table>
<thead>
<tr>
<th>ino</th>
<th>color</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12345</td>
<td>red</td>
<td>A</td>
</tr>
</tbody>
</table>

transaction 31 completes

/q
Good Bye !!!

$2^D$
script done on Wed May 28 18:55:50 1986