ECRINS/86: An Extended Entity-Relationship Data Base Management System and its Semantic Query Language

FALQUET, Gilles, LEONARD, Michel Paul, JUNET, Marc

Abstract
We propose a DBMS the data model of which is an extension of the Entity-Relationship model of Chen. The main extensions include the specialization and generalization concepts of Smith & Smith and the possibility to define relationships between relationships. In addition we extend the concept of a role in a relationship, to a multi-valued role which enables to associate a set of tuples performing the same function in a relationship. A query language has been designed in order to deal with the enhanced semantic capabilities of the data model supported by the ECRINS/86 system.

Reference

Available at: http://archive-ouverte.unige.ch/unige:46595
Disclaimer: layout of this document may differ from the published version.
ECRINS/86 : An Extended Entity-Relationship Data Base Management System
and its Semantic Query Language

Marc Junet, Gilles Falquet, Michel Leonard

University of Geneva, Centre Universitaire d'Informatique
12, rue du Lac, CH - 1207 Geneva, Switzerland

ABSTRACT : we propose a DBMS the data model of
which is an extension of the Entity-Relationship
model of Chen. The main extensions include the
specialization and generalization concepts of
Smith & Smith and the possibility to define
relationships between relationships. In addition
we extend the concept of a role in a
relationship, to a multi-valued role which
enables to associate a set of tuples performing
the same function in a relationship. A query
language has been designed in order to deal with
the enhanced semantic capabilities of the data
model supported by the ECRINS/86 system.

1. INTRODUCTION

ECRINS/86 is a DBMS (Data Base Management System)
the model of which is an Extended
Entity-Relationship Model. This paper is
narrowed to the description of the extensions of the
E-R Model proposed by Chen [CHEN76] and to
the semantic query language we developed to deal
with the ECRINS/86 system. A more complete
description of the system may be found in
[JUNET86]. In addition to the concepts proposed by
Chen we included the generalization and
specialization concepts developed by Smith &
Smith [SMITH77]. We also extended the role
concept of an entity in a relationship. A role
in the E-R Model is the function that an entity
performs in a relationship; through one role only
one tuple of an entity relation can be related to
one tuple of a relationship relation. We
introduce the new concept of multi-valued role
which enables to associate a set of tuples
performing the same function in a relationship.
Furthermore in ECRINS/86 a role in a relationship
relation may be performed either by an entity or
another relationship relation (the original E-R
Model was criticized for its lack of capability
to express such relationships [SCHEUER79]).
In addition a new graphical representation adapted
to the Extended E-R Model of ECRINS/86 is
proposed. Obviously the ECRINS/86 system
validates automatically all the inherent
integrity constraints related to the data model
implemented. The description of those integrity
constraints can be found in [JUNET86]. They
mainly concern : a) the domain of the attributes,
b) the keys of the relations, c) the existence
dependencies, d) the cardinality and e) unknown
values.

In section 2 only the new concepts of the data
model are presented in order to explain and
illustrate the particularities of the query
language. Section 3 describes some features of
this query language. These are : 1) the
definition and use of extended relation obtained
by joining tuples of different relations
associated through relationship relations,
2) the handling of non-first normal form
relations appearing when multi-valued roles
are defined in a relationship relation, 3) the
deduction of an extended relation containing all
the attributes of a query.

2. THE EXTENDED E-R MODEL IMPLEMENTED WITH THE
DBMS ECRINS/86

In order to point out some of the modeling
facilities offered by the Extended E-R Model of
ECRINS/86, we will first model an example
concerning an airline company with the n-ary
relational model of Codd [COOD70].

2.1 A n-ary Relational Modeling Example

Let us consider an airline company made up of the
following relations and attributes (the keys of
each relation are underlined).

FLYING-STAFF (NAME, ACE, SALARY, JOB-CLASS)
PILOT (PNAME, LICENCE-NUMBER, LICENCE-TYPE)
OPERATOR (ONAME, JOB-HISTORY)
STEWARDESS (SNAME, YEARS-OF-SERVICE)
CABIN-CREW (CREW-NUMBER, CHIEFNAME, COP1-
PNAME, COP2PNAME, ONAME, NBFLIGHT)
-----------------------
AIRPORT (AIRPORT-NAME, CITY)
PLANE (PLANE-NUMBER, KIND-OF-PLANE, NB-SEAT)
FLIGHT (FLIGHT-NUMBER, CREW-NUMBER, SNAME1,
SHANE2, SNAME3, SNAME4, DEP-AIRPORT-
NAME, ARR-AIRPORT-NAME, PLANE-NUMBER,
DEP-HOUR, ARR-HOUR)

The relations PILOT, OPERATOR and STEWARDESS are
specializations of the relation FLYING-STAFF.
With the n-ary relational model 4 relations need
to be defined in order to model this
specialization concept. The data base designer
must also be aware of all the integrity
constraints which are related to this structure
(i.e. deleting a tuple of PILOT must trigger the
deletion of the corresponding tuple in relation
FLYING-STAFF).
Within the relation CABIN-CREW we observe two different keys and the following functional dependencies:

CREW-NUMBER -> CHIEFPNAME, COP1PNAME, COP2PNAME, ONAME, NBFLIGHT
CHIEFPNAME, COP1PNAME, COP2PNAME, ONAME -> CREW-NUMBER, NBFLIGHT

Nevertheless a chief pilot and a co-pilot are both pilots. In other words the attributes CHIEFPNAME, COP1PNAME and COP2PNAME have the same domain and semantic as the attribute PNAME of the relation PILOT. The n-ary relational model is not appropriate to explicitly express this correspondence between CHIEFPNAME, COP1PNAME, COP2PNAME with PNAME.

The problem described for the relation CABIN-CREW is identical for the relation FLIGHT. But concerning this relation a new problem is due to "undefined" unknown values ("undefined" and "nothing" unknown values were introduced by Abrial in [ABRIAL74] and discussed by Tjoa [TJOA79] concerning their implications within the E-R Model). Considering the fact that a flight may need only two stewardesses, the key attributes SNAME3 and SNAME4 will have "undefined" unknown values. But with the n-ary relational model "undefined" unknown values are prohibited for key attributes.

2.2 Data Model and Definitions

The Extended E-R Model of the ECRINS/86 system takes care of the problems we briefly described in the airline company example. We shall now define the different concepts used in our Extended E-R Model in order to show how the airline company example can be modeled with ECRINS/86. In the appendix A, a definition of this example with the Data base Definition Language (DDL) of the ECRINS/86 system is proposed.

2.2.1 Regular Entity Relation

A regular entity relation is composed by a set of attributes and a primary key (PK) is defined with a subset of attributes. For example: the n-ary relations FLYING-STAFF, AIRPORT and PLANE can be defined as regular entity relations.

2.2.2 Sub-relation

The term of sub-relation is used to define the generalization and specialization concepts [SMITH77]. In ECRINS/86 we restricted the implementation of these concepts to a "tree generic hierarchy". According to this restriction a specialization can be considered as a subset of a relation and we shall call it sub-relation.

A sub-relation SR is defined from one and only one relation R (called the SR generic relation). A tuple of SR is a tuple of R, and R attributes may be viewed as SR attributes. The PK of SR is the PK of R. A generic relation may be of any kind (entity, relationship, sub-relation). Furthermore a SR may have own attributes. For example: the n-ary relations PILOT, OPERATOR and STEWARDESS can be defined as sub-relations of the regular entity relation FLYING-STAFF.

Obviously the ECRINS/86 system validates all the inherent constraints due to the specialization and generalization concepts we implemented.

2.2.3 Relationship Relation

In ECRINS/86 no distinction is made between weak and regular relationship relations. Both are handled in the same way since we allow any kind of relation playing a role in a relationship relation. A relationship relation RS is a relation which is defined over n relations: Rf1, Rf2, ..., Rfm; they are called the RS reference relations (reference relations may not be distinct). A reference relation may be either a regular entity relation, a weak entity relation, a sub-relation or a relationship relation.

A role of a reference relation in a relationship relation is the function that it performs in the relationship. Each reference relation may perform m distinct roles in a relationship: r1*, r2*, ..., rm*. A role is defined as a simple role when a single tuple of a reference relation is associated to one tuple of a relationship relation. A role is a multi-valued role when a set of tuples is associated to one tuple of a relationship relation. Multi-valued roles are distinguished from simple roles with a superscript "*" (i.e. r1* is a multi-valued role) and the maximum number of tuples which can be associated is the degree of the multi-valued role. In the ECRINS/86 system a role may be declared as "not-mandatory" (the "mandatory" concept of a role is similar to "mandatory automatic set" in [CUVASLY1]). It is then possible to create tuples in a relationship relation with "undefined" unknown values through those roles.

The set of attributes of a relationship relation is composed by the PK of the reference relations, plus own attributes. For each role that is performed by a reference relation in a relationship, the PK of the reference relation is "exported" to the relationship relation. This set of "exported" attributes can be considered as the PK of the relationship relation. The PK "exported" through a multi-valued role becomes an higher order object (see definition in section 3.3.1), noted PK*, in the relationship relation. It may be possible to define a secondary key to a relationship relation representing the aggregation of all the "imported" attributes.

Example:

The n-ary relation CABIN-CREW can be defined as a relationship relation with 2 reference relations: PILOT and OPERATOR. PILOT performs the simple role CHIEF-PILOT and the...
multi-valued role CO-PILOT* of degree 2 in
the relationship. OPERATOR performs a simple
role OPERATOR-OF-CREW. The PK of CABIN-CREW
is composed by the attributes : NAME (as a
chief-pilot), NAME* (as 2 co-pilots), NAME
(as an operator). The secondary key is the
attribute CREW-NUMBER.

Through the multi-valued role CO-PILOT, two
pilots may be associated to one cabin-crew.
Without the concept of multi-valued role a
database designer must declare two different
roles, CO-PILOT-1 and CO-PILOT-2, to express the
fact that a cabin-crew may have two co-pilots. In
this case, a pilot does not play the same role in
the relationship. He is either a first
(CO-PILOT-1) or a second (CO-PILOT-2) co-pilot.

In the same way the n-ary relation FLIGHT can
be defined as a relationship relation, the
reference relations of which are : CABIN-CREW
(role : CREW-OF-FLIGHT), STEWARDESS (role :
STEWARDESS-OF-FLIGHT*), AIRPORT (roles :
DEP-AIRPORT, ARR-AIRPORT), PLANE (role :
PLANE-OF-FLIGHT). The degree of the role
STEWARDESS-OF-FLIGHT* is 4, but by defining
this role as "not-mandatory", only 2
stewardesses may appear in one flight.

When a reference relation RF performs one
multi-valued role and/or many roles in a
relationship relation RS, it may be possible to
declare RF as mono-reference. This enables to
insure that a tuple rf of RF may not be
associated in a tuple rs of RS more than once
(mono-tuple association). Otherwise there is a
multi-tuple association.

Example :

PILOT is mono-reference in CABIN-CREW
considering that a tuple p of PILOT may
not be in a tuple cc of CABIN-CREW, a
chief-pilot and a co-pilot. AIRPORT is
multi-reference in FLIGHT considering that a
tuple a of AIRPORT may be in a tuple f of
FLIGHT, the departure and the arrival
airport.

To each role it may be possible to declare a
maximum cardinality parameter (maxcard) which
indicates the maximum of RS tuples which can be
related to one tuple of a reference relation
through this role. This parameter maxcard enables to declare all the different kinds of
mapping of a relationship relation.

Obviously the ECRINS/86 system validates
automatically all the inherent constraints due to
the relationship relations (i.e. existence
dependencies, cardinality constraints, ...).

2.2.4 Weak Entity Relation

We define a weak entity relation as a relation in
which the existence of a tuple depends upon the
existence of a specific tuple of a reference
relation RF. In the same way as relationship
relations, a reference relation may be of any
kind. The PK of a weak entity relation is
composed by the PK of its reference relation RF
and by other supplementary attributes. There is a 1:M mapping between RF and the weak entity
relation.

Example :

Let consider a new relation QUALIFICATION as
a weak entity relation which reference
relation is OPERATOR. Its PK is composed by
the key attribute of OPERATOR (NAME) and a
supplementary attribute QUAL-NUMBER (which is
a serial number used to distinguish the
various qualifications of one operator). Own
attributes of QUALIFICATION are an
explanation of the qualification
(EXPLANATION) and the number of years the
operator practiced it (YEARS-OF-PRACTICE).

2.2.5 Graphical Representation

As soon as a data schema contains several
relations of various kinds, the use of a
graphical representation helps in the
understanding and the communication. Because of
the features of the Extended E-R Model
implemented with the ECRINS/86 system, we defined
new conventions for a graphical representation.

The set of relations is mapped onto a graph where
single nodes correspond to regular (thin) and
weak entity relations (thick), thin double nodes
to relationship relations. Thin edges correspond
to roles and thick edges to links between a weak
entity relations and its reference relation. Thin
edges are directed from a relationship relation
to its corresponding reference relations, the
name, the cardinality, the minimum and maximum
degree of a multi-valued role are put near to it.
A single arrow is used for simple roles, while
double arrows represent multi-valued roles.

Sub-relations which have the same generic
relation are mapped onto single nodes put inside
the node of their generic relation. A symbol
representing the exclusive or (V) is used to show
the different sub-relations of a relation.

We introduce 4 new sub-relations to the airline
company example in order to show how it is
possible to represent a sub-relation the generic
relation of which is another sub-relation or a
relationship relation. Let consider that an
operator may be either a student (sub-relation
STUDENT) or a diplomed operator (sub-relation
DIPLOMED) and a cabin-crew either active
(sub-relation ACTIVE) or inactive (sub-relation
INACTIVE). In a tuple of relation CABIN-CREW only
one diplomed operator may be involved and in a
tuple of relation FLIGHT only an active
cabin-crew.
3. THE ECRINS/86 QUERY LANGUAGE

3.1 General Structure

The basic structure of the query language is a "query block" similar to what can be found in common languages such as SQL. A query block is made up of a target attributes list specifying the attributes to be output, a "from" clause specifying the relations to use in the query and a "where" clause giving the selection condition. The following sections describe some aspects of the query language related to the structure and the semantic of the data model implemented with ECRINS/86; these are: extended relations, nested relations structures and deduction of extended relations.

3.2 Extended Relations

The "from" clause of a query may contain, apart from entity and relationship relations, a list of expressions defining derived relationship relations called extended relations. The use of extended relations, expressed in a simple form, allows to withdraw from the "where" clause the conditions that are only used to associate tuples of different relations.

A linear-extended relation expression is an expression of the form:

\[ ER = \text{Rel}_1 \text{ of } \text{Rel}_2 \text{ of } \ldots \text{ of } \text{Rel}_n, \]

where the Rel_i's are entity or relationship relations.

The corresponding derived relation is the set of all tuples \(<t_1, ..., t_n>\) where \(t_i\) is a tuple of Rel_i and \(t_i\) is associated to \(t_{i+1}\) through the simple role \(r_{ij} (0 < i < n)\) (the semantic of associations through multi-valued roles is described in section 3.3). Let us also define first(ER) as Rel_1 and last(ER) as Rel_n.

Example:

PLANE of FLIGHT of ARR-AIRPORT of AIRPORT

is the extended relation associating a flight to its plane and its arrival airport. Note the absence of role specification between PLANE and FLIGHT since PLANE performs only one role in the relationship relation FLIGHT.

A tree-extended relation is an expression of the form:

\[ ER = (ER_1 \text{ of } r_{11} \text{ of } ER_2 \text{ of } r_{22} \ldots \text{ of } ER_m \text{ of } r_{mm}), \]

where the ER_j's are extended relations and \(r_{ij}\) associates last(ER_j) with first(ER_j) (ER_m may not be a tree extended relation). The corresponding derived relation is the set of all tuples \(<<t_1, ..., t_{kl}>, ..., <t_m, ..., t_{km}>\) such that \(t_1, ..., t_{kl}\) belongs to ER_1 \((0 < i < m+1)\) and \(t_{kj}\) is associated through role \(r_{ij}\).

In this case last(ER) is defined as last(ER_m) and first(ER) is undefined.

Example:

(OPERATOR of CABIN-CREW, AIRPORT of ARR-AIRPORT) of FLIGHT of PLANE

Finally a cyclic-extended relation is an expression of the form:

\[ ER = ER_1 \text{ of } (r_{11} \text{ of } ER_2 \text{ of } r_{22} \ldots \text{ of } \ldots \text{ of } r_{mm} \text{ of } ER_1), \]

where the ER_j's are extended relations and \(r_{ij}\) associates last(ER_j) to first(ER_j) (first(ER_j)) and \(r_{ij}\) associates last(ER_j) (last(ER_j)) to first(ER_j).<connect> is one of the connectors: and, or, and. The corresponding extended relation is the set of tuples \(<<t_1, ..., t_{kl}>, ..., <t_m, ..., t_{km}>\) such that \(t_1, ..., t_{kl}\) belongs to ER_1 \((0 < i < m+1)\) and:

a) and

\(<C1> t_{1j}^{k1} \text{ is associated to } t_{2j}^{k2} \ldots \text{ through } r_{11} \text{ and } t_{2j}^{k2} \text{ is associated to } t_{3j}^{k3} \ldots \text{ through } r_{22} \text{ and } t_{3j}^{k3} \text{ is associated to } t_{4j}^{k4} \ldots \text{ through } r_{33} \text{ and } t_{4j}^{k4} \text{ is associated to } t_{4j}^{k4} \ldots \text{ through } r_{44}.\)

b) or

\(<C1> \text{ and } t_{3j}^{k3} \text{ is null and there is no tuple } t_{3j}^{k3} \text{ of ER_3 associating } t_{1j}^{k1} \text{ and } t_{4j}^{k4} \text{ or }\)

\(<C2> \text{ and } t_{2j}^{k2} \text{ is null and there is no tuple } t_{2j}^{k2} \text{ in ER_2 associating } t_{1j}^{k1} \text{ and } t_{4j}^{k4}.\)
c) **diff**

Cl and t₃ is null and there is no tuple t₃' of ER₃ associating t₁ and t₄.

**first(ER)** is defined as first(ER₁) and **last(ER)** as last (ER₄).

**Example:**

AIRPORT of (DEP-AIRPORT of FLIGHT or ARR-AIRPORT of FLIGHT) of CABIN-CREW.

Corresponds to the relation associating each flight to its cabin-crew to its departure or arrival airport.

As the same relation may appear many times in an extended relation expression, an explicit renaming can be used to distinguish different occurrences. The new names can then serve as a qualification for attributes.

**Example:**

"Find the city of arrival, the city of departure, the operator's and chief-pilot's names of every flight"

select CITY of AR, CITY of DEP, NAME of OPERATOR, NAME of PILOT
from (PILOT of CHIEF-PILOT, OPERATOR) of CREW,
DEP : AIRPORT of DEP-AIRPORT,
AR : AIRPORT of ARR-AIRPORT of FLIGHT.

Note the nesting of the tree extended relations.

### 3.3 Nested Extended Relations

The presence of multi-valued roles in the data model leads naturally to non-first normal form extended relations. Non-1NF relations, [ABIT85] [FISCH85] are relations which tuples are defined on atomic values and/or (non-1NF) relations. This kind of structure is what one intuitively expects when associating relations through multi-valued roles. The expected meaning of "STEWARDESS of FLIGHT of PLANE" is a set of tuples <f,p> where f is a flight, p is its plane and e is the set of stewardesses (i.e. a relation) of this flight. Non-1NF relation also arise when considering weak entity relations. Since many tuples of a weak entity relation are associated to one tuple of the reference relation, they can be considered as a relation nested in their reference tuple. For example:

**OPERATOR** (NAME, EXPERIENCE-HISTORY, QUALIFICATION*)
with QUALIFICATION (QUALIFICATION NUMBER, EXPLANATION, YEARS-OF-PRACTICE). Nested relations are evidently not limited to one level.

#### 3.3.1 Nested Joins

Let us first define a nested relation scheme as a set of attributes and schemes (called higher order objects). For example:

**FLIGHT** (FLIGHT-NUMBER, CREW-NUMBER, STD*)
with the scheme
STD(NAME, YEARS-OF-SERVICE)

is a nested relation scheme (with one level of nesting).

A (nested) relation over a nested relation scheme (A₁,...Aₘ,Y₁,...Yₙ*) (a * follows each higher order object) is then recursively defined as a set of tuples <a₁,...,aₘ,Y₁,...,Yₙ*> where aᵢ is a value taken from the domain of Aᵢ (1 ≤ i ≤ m) and yᵢ is a (nested) relation on scheme Yᵢ (1 < j < n).

Given two schemes:

B = (X₁*,...,Xₚ*), S = (Z₁*,...,Zₚ*)

with

K₁* in X₁*, K₂* in K₁*, ..., Kₚ* in Kₚ*, A in Kₚ*

and

L₁* in U₁*, L₂* in L₁*, ..., Lₚ* in Lₚ*, B in Lₚ*

and two relations:

I on R and J on S,

the nested equi-join of I and J on A and B used in ECRINS/B6 (noted I*<A=B>*J) is recursively defined as:

1. if n = m (i.e. the two attributes are at the same level)

   1.1 if A is in R and B is in S then
   I*<A=B>*J = I< A=B> J
   (the standard equi-join of "flat" relations)

   1.2 else
   I*<A=B>*J = { t / there exists u in I, v in J such that
   t[X₁,...,Xₚ*] = u[X₁,...,Xₚ*] and
   t[Z₁,...,Zₚ*] = v[Z₁,...,Zₚ*] and
   t[(Y₁U₁)*] = u[(Y₁)*] for A=B > v[(U₁)*] ≠ Ø }.

2. if n is different from m (say m < n) then

   I*<A=B>*J = I*<A>B>*aug(J) where p = n - m
   and aug(J) (the structural augmentation of J) is a relation defined on (Z₁*,...,Zₚ*)
   and composed of only one tuple t with
   t[(Z₁,...,Zₚ*)] = J.

In the appendix B, two examples of nested equi-join are given.

A nested theta-join may be defined in a similar way, as well as a nested set-theta-join
with set comparison operators between higher order objects replacing the join attributes.

#### 3.3.2 Extended Relations with Multi-Valued Roles and Set Expressions

The nested extended relation corresponding to a linear extended relation expression with multi-valued roles is obtained by taking the nested equi-join of all the relations with the PK attributes representing the different roles used as join attributes. If there's no multi-valued role the resulting relation is equal to the linear extended relation defined in section 3.2. Nested tree and cyclic extended relations are defined in a similar way as tree and cyclic extended relations but with nested equi-joins used to associate the different (nested) extended relations.
Examples:

1. STEWARDESS of FLIGHT of PLANE is interpreted as:
   STEWARDESS*(NAME=SNAME)*FLIGHT*
   <PLANE-NUMBER=PLANE-NUMBER>*PLANE
   with FLIGHT being a nested relation defined on
   (FLIGHT-NUMBER, CREW-NUMBER,
   DEP-AIRPORT-NAME, ARR-AIRPORT-NAME,
   DEP-HOUR, ARR-HOUR, (SNAME)*).

2. Let TICKET be a relationship relation which reference relation is FLIGHT performing a multi-valued role, then
   STEWARDESS of FLIGHT of TICKET is interpreted as:
   STEWARDESS*(NAME=SNAME)*FLIGHT*
   <FLIGHT-NUMBER=FLIGHT-NUMBER>*TICKET
   which associates to a ticket a set of flight and to each one of those flights a set of stewardess.

   The connectors containing, included in, equals, etc. appearing in an extended relation expression give rise to set-theta-joins in the computing of the extended relation.

   Example:
   "Find all the flight numbers of flights needing all the stewardess of flight number 102".
   select FLIGHT-NUMBER of F1
   from F1 : FLIGHT of STEWARDESS containing
   STEWARDESS of F2 : FLIGHT
   where FLIGHT-NUMBER of F2 = 102.

   Two additional operators : one and set [MARKOW83] can be applied to convert a multi-valued role to a simple role and vice-versa.

   Example:
   "Find the name of the co-pilots of any cabin crew who have a type B licence".
   select NAME of PILOT
   from PILOT of one CO-PILOT of CABIN-CREW
   where LICENCE-TYPE = "B".

3.4 Deduction of Extended Relations

3.4.1 Qualification Expressions

A qualification is an expression taking the same form as an extended relation. A qualified attribute [MGREG85] is an attribute A followed by a qualification ER such that A is an attribute of first(ER).

The qualifications are used in a query block to specify to which extended relation a set of attributes belongs. Given a set of qualifications ER1,...,ERk, the corresponding extended relation is defined as then minimal extended relation "covering" the ERi's. If more than one such extended relation exists then the qualifications are said to be ambiguous. Thus the qualifications determine the semantic connection between attributes of the query.

Example:
select NAME of DIPLOMED,
   KIND-OF-PLANE of PLANE
is interpreted as
select NAME of DIPLOMED,
   KIND-OF-PLANE of PLANE
from DIPLOMED of ACTIVE of FLIGHT of PLANE
while
select NAME of PILOT, KIND-OF-PLANE of PLANE
is ambiguous since PILOT and PLANE can be associated through CHIEF-PILOT or CO-PILOT.

3.4.2 Abreviated Qualifications

ECRINS/86 offers the possibility to abbreviate the qualification of an attribute as long as no ambiguity appears. An abreviated (or incomplete) qualification is said to be unambiguous if it can be expanded to only one qualification.

Example:
NAME of OPERATOR of PLANE
will be expanded to
NAME of DIPLOMED of ACTIVE of FLIGHT of PLANE.

4. CONCLUSION

The data model we implemented with the ECRINS/86 system is powerful enough to take care of the main integrity constraints of a complex data schema. The main advantage of a DBMS such as ECRINS/86 is due to its capability to implement a data structure very quickly without developing wearisome validation programs. The query language allows the user to access the database in a simple way. The construct of the language are closely related to the data model and thus provide a coherent interface to the data base. Actually the ECRINS/86 system runs on computers such as UNIVAC 1100/60, VAX-780 (VMS+UNIX systems). PRIME-750. Personal Computer running with MS/DOS, SUN, ... A first version of ECRINS/86, restricted to binary relationship relations [LEON85], has been installed in some universities in Switzerland and in Europe, as well as in private compagnies. We also used this version to manage the meta-base of two DBMS implemented at the C.U.I. The first one is PIREE used to store data on economics, the second one is FARANDOLE used in data analysis [SNEILL86]. The query language is under development; nevertheless a subset of it, including extended relations and automatic extended relations deduction, has been implemented for the PIREE DBMS. Since ECRINS/86 is not a front-end for another DBMS such as Ariel [MGREG85] the execution of queries can take advantage of physical data structures well suited for implementing an extended E-R Model.
ACKNOWLEDGEMENT

The authors would like to thank all the members of the Data Base Group in the C.U.I. who contributed to the implementation of the system. Special thanks go to R. Tschopp, A. Galland for their earlier contribution to the project, to F. Bodard for his valuable comments and to E. Kohl for her precious collaboration for the page-setting of this paper.

REFERENCES


Appendix A: The "airline company" example defined with the ECRINS/86 Data Definition Language.

This definition includes all the extensions added throughout this paper.

structure AIRLINE-COMPANY
begin
declare regular entity relation FLYING-STAFF
key is
    NAME char (36);
with properties
    AGE integer (12:75);
    SALARY real (1000:20000);
    JOB-CLASS word generic (PILOT, OPERATOR, STEWARDESS);
    MARITAL-STATUS word (SINGLE, MARRIED, DIVORCED, WIDOWED);
if not SINGLE then
    YEARS-OF-MARRIAGE integer (1900:2100);
    PLACE-OF-MARRIAGE char;
endif
end-declare

declare sub-relation PILOT
with properties
    LICENCE-TYPE char;
secondary key is
    LICENCE-NUMBER integer;
end-declare

declare sub-relation OPERATOR
with properties
    JOB-HISTORY char;
    STATUS word generic (STUDENT, DIPLOMED) mandatory;
end-declare

declare sub-relation STUDENT
with properties
    YEARS-OF-STUDY integer (1:5);
end-declare

declare sub-relation DIPLOMED
with properties
    KIND-OF-DIPLOMA char;
end-declare

declare sub-relation STEWARDESS
with properties
    YEARS-OF-SERVICE integer (0:50);
end-declare

declare weak entity relation QUALIFICATION
reference is OPERATOR with maxcard 5
key is
    QUAL-NUMBER rank;
end-declare

[-265-]
with properties
EXPLANATION char;
YEARS-OF-PRACTICE integer (0:50);
end-declare

declare relationship relation CABIN-CREW
association of
PILOT mono-reference
(CHIEF-PILOT) multi-valued of degree 2
with maxcard 15
DIPLOMED with maxcard 15
with properties
NBFLIGHT integer (0:10000);
CREW-STATUS word generic (sr INACTIVE
or ACTIVE);
secondary key is
CREW-NUMBER integer (100:200);
end-declare

declare sub-relation INACTIVE
with properties
REASONS char;
end-declare

declare regular entity relation AIRPORT
key is
AIRPORT-NAME char (24);
with properties
CITY char (24);
end-declare

declare regular entity relation PLANE
key is
PLANE-NUMBER integer;
with properties
KIND-OF-PLANE char mandatory;
NB-SEAT integer (12:500) mandatory;
end-declare

declare relationship relation FLIGHT
association of
STEWARDESS multi-valued of degree 4
not mandatory with maxcard 100
ACTIVE with maxcard unknown
PLANE with maxcard 25
AIRPORT (DEP-AIRPORT) with maxcard 100
(ARR-AIRPORT) with maxcard 100
with properties
DEP-HOUR real;
ARR-HOUR real;
secondary key is
FLIGHT-NUMBER integer (100:1000);
end-declare

declare relationship relation TICKET
association upon
FLIGHT mono-reference
multi-valued of degree 3 not mandatory
with maxcard unknown
with properties
DATE-OF-ISSUE integer mandatory;
PRICE real mandatory;
PASSENGER-NAME char (36) mandatory;
secondary key is
TICKET-NUMBER integer;
end-declare

Appendix B : Examples of nested equi-joins

Example 1 :

<table>
<thead>
<tr>
<th>FLIGHT</th>
<th>GROUP-OF-STEWARDESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLT-NO.</td>
<td>STD</td>
</tr>
<tr>
<td>101</td>
<td>name1</td>
</tr>
<tr>
<td></td>
<td>name3</td>
</tr>
<tr>
<td>103</td>
<td>name2</td>
</tr>
<tr>
<td>104</td>
<td>name5</td>
</tr>
</tbody>
</table>

Example 2 :

<table>
<thead>
<tr>
<th>STEWARDESS</th>
<th>NAME</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>name1</td>
<td>21000</td>
<td></td>
</tr>
<tr>
<td>name2</td>
<td>19000</td>
<td></td>
</tr>
<tr>
<td>name3</td>
<td>22000</td>
<td></td>
</tr>
<tr>
<td>name4</td>
<td>21500</td>
<td></td>
</tr>
<tr>
<td>name5</td>
<td>27000</td>
<td></td>
</tr>
<tr>
<td>name6</td>
<td>22000</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FLIGHT*&lt;NAME=SNAME&gt;*STEWARDESS:</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLT-NO.</td>
</tr>
<tr>
<td>101</td>
</tr>
<tr>
<td>102</td>
</tr>
<tr>
<td>103</td>
</tr>
<tr>
<td>104</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>