An Approach to Eliciting the Semantics of Relational Databases

M.M.Fonkam W.A.Gray

Dept. of Computing Maths, University of Wales College of Cardiff,
Cardiff CF2 4AG, email : mmf@uk.ac.cf.cm.

Abstract. Relational database systems are currently the most dominant in both centralised and distributed database environments inspite of the many limitations of these systems. A fundamental weakness of these systems is that the logical structure of the data of their DBs which must be exploited by their users is usually buried in the DB schemas, application programs and in the minds of designers and programmers. The vast number of casual and inexperienced users currently employing these DBs for their day to day applications makes it imperative for their logical structure to be made explicit and readily available in some easily assimilated form.

A number of algorithms have been proposed for this reverse modelling activity which essentially generate a conceptual model from a given relational DB schema. In this paper we make a comparative study of three representative previous algorithms and then present a new improved and more general algorithm based on these previous attempts.

1 Introduction

The current dominance of the relational model in the commercial scene coupled with its lack of facilities by which users can easily display and understand the semantics of its databases has led to research into ways of converting the schemas (intension) of these databases into conceptual schemas using one of the semantic data models, usually the Entity-Relationship (ER) model or a semantically richer variant of it. The result of this research has been a number of algorithms, each prescribing a set of rules by which the implied semantics of a relational schema can be extracted and re-expressed using a semantic data model (SDM) where such semantics are made explicit, [DAV88, JOH89, KAL91, NAV88]. The input to these algorithms is the relational schema of a database possibly enhanced with extra information. The ER-models are heavily used as conceptual models since one of their attractive features is their ability to express the contents of a DB in a form quite close to a user's perception. Conceptual models such as the ER-model, explicitly maintain the relationships of the database and some of these models make provisions for querying such relationships in much the same way as one would query the data.

There are four main sources of the semantics of a given relational DB, namely; the *database schema;* the *application programs* written for specialised purposes; the *explicit integrity constraints* and *users of the designer and application programmer* categories. All four sources may be relevant for producing the conceptual schema of a relational DB that captures its complete semantics. This paper is concerned with the *inherent or structural semantics* of a relational DB and as such only those explicit integrity constraints that are also structural constraints will be considered. Our own work, like previous works in this area does not make use of the semantics embedded in application programs.

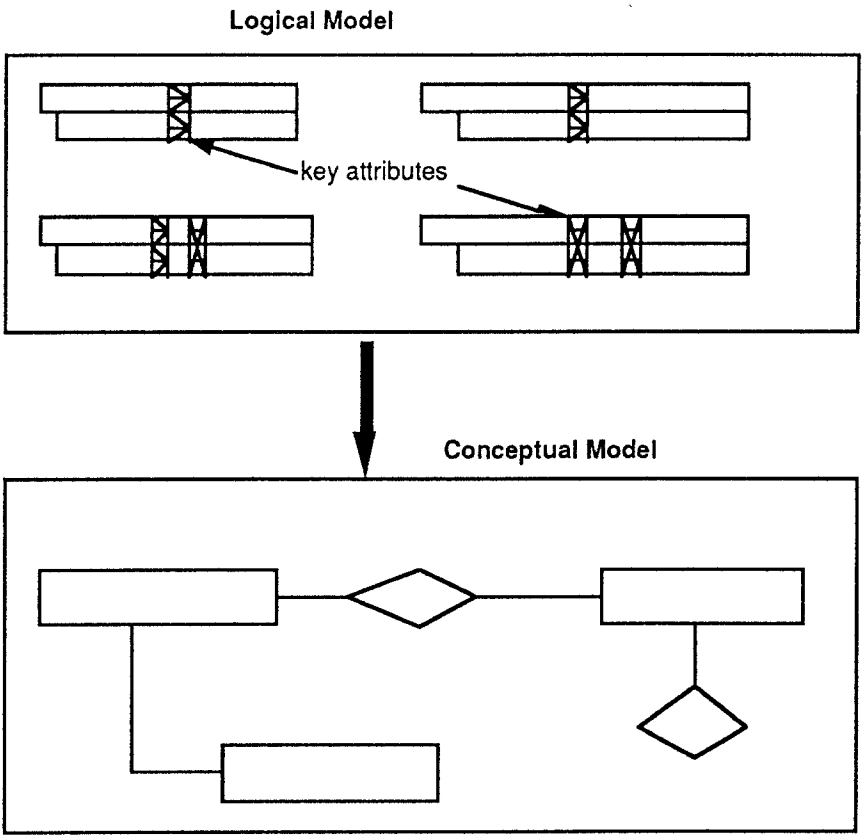

Fig. 1. Reverse Modelling of A Relational Schema into a Conceptual Model

A notable problem of relational systems is that they represent relationships only implicitly through matching field values in relations and the majority of systems make no provision for allowing users to discover such relationships. The vast range and number of users now currently employing relational DBs justifies then the need for a reverse modelling process whereby the intension of the database can be mapped into a conceptual schema which shows its relationships and structure explicitly. This process of reverse

modelling is graphically illustrated in Fig.1, where the logical model shows the tables of a relational schema being mapped into a more powerful pictorial representation. The need for such reverse modelling or reverse engineering is even greater in a Multidatabase System (MBS) [LIT88] environment where users do not only have access to their own local DBs but can also access remote independent DBs in a network of interconnected databases. Without such a tool in these environments, most users would find it practically impossible to exploit the wealth of data available to them.

Many algorithms have been proposed in the literature for converting relational database schemas into conceptual schemas under the ER model or some variant of the ER model [DAV88,JOH89,KAL91,NAV88]. While most of these algorithms share a number of features in common, we have found three distinct algorithms that together embody the essential features of all the previous algorithms but which each adopts a significantly different approach from any other algorithm. Each of these algorithms suggests some significant concepts to be embodied in a more generic translation algorithm. These are Davis and Arora's algorithm [DAV88] which translates both the structure and the explicit behaviour of the model, Navathe and Awong's algorithm [NAV88] which classifies the relations and attributes and incorporates a preprocessing step for renaming or changing the roles of attributes of relations, and Kalman and Johanneson's algorithm [JOH89] which also classifies the relations and attributes of the schema but only uses inclusion dependencies.

Most of the previous algorithms acknowledge the tight interaction needed with an experienced user (i.e. a user who understands the intended semantics of the DB) to alleviate certain semantic ambiquities about the data. Such an experienced user is typically assumed to be a specialist in DB design who possesses the relevant knowledge about the present DB and has the skills needed to use the algorithm in developing a conceptual model. A paramount objective of these algorithms is to *automate* the process as far as possible. The major limitation of these algorithms, however, is their *lack of generality* which means that for certain DB schemas they will miss out some very vital semantics that need to be made explicit in the conceptual model. From a detailed study of these algorithms, we have identified a number of their limitations, some of which have been studied by Kalman in [KAL91]. Then by extracting the important concepts from each of these algorithms and introducing some new concepts a new more general and highly automated algorithm has been built that alleviates many of these limitations. Our implementation produces a graphical output of the conceptual model which shows the relationship between the conceptual model obtained and the underlying relational model, thus enabling the user to directly frame DB queries from knowledge of the conceptual model alone. This linkage is an innovation of this algorithm which assists users by improving their perception of the DB semantics and its realisation in the relational model.

In the next section of this paper, we make a detailed comparison of the three main algorithms mentioned above highlighting their relevant contributions as well as their major limitations. This comparison is based on a number of criteria which we considered relevant for such translation algorithms. In Section 3, we briefly introduce the Entity-

Category-Relationship(ECR) model which we use to represent the semantic data model at the conceptual level. Section 4 considers modelling within the relational context. Sections 3 and 4 serve to give the reader a better understanding of the reasoning behind the new algorithm. Section 5 presents our new algorithm, first through examining the general rules that have been extracted from the previous algorithms, then by presenting new relevant rules. Section 6 is concerned with its implementation and explores the types of interactions needed with the user to extract further semantics.We also discuss the graphical output that represents the conceptual model produced by the new algorithm and then illustrate the technique adopted in this graphical output to assist the user gain an understanding of the connection between this conceptual model and the underlying relational schema. In the conclusions, Section 7, we examine the generality of our new algorithm, its contribution and present some directions for future work.

2 A Comparative Study of Existing Algorithms

Six main criteria have been used as a basis for our comparison. These are briefly introduced in this section. In this comparison process we will denote by Algorithm 1 Davis and Arora's algorithm, by Algorithm 2 Navathe and Awong's algorithm and by Algorithm 3 Kalman and Johanneson's algorithm.

a). the equivalence of the models - to what extent is the conceptual model a reflection of its relational model? As pointed out in Tsichristis and Lochovsky [TSI82], this equivalence can be shown if an inverse mapping can be found for converting the conceptual model back to the original relational model. The authors of Algorithm 1 actually showed in their work, [DAV88], how an inverse mapping could be found for mapping from the conceptual model back to the original relational model. An inverse mapping can also be easily found for Algorithms 2 and 3. In fact, this equivalence derives from the easy mapping of ER models to relational models, thus once a good ER model has been derived then the equivalence of these models is ensured[ELM89].

b). the inclusion and handling of subtype/supertype semantics - One main weakness of the relational model is its lack of an explicit means to model semantics of the generalisation/specialisation type. There is no generally agreed way of modelling such semantics implicitly in relational systems; some designers may choose to simply create separate base relations sharing common key attributes for the subtype and supertype, while others may employ views to model these semantics. The view approach [RAM89, RAM91] seems more attractive since to some albeit, limited extent, it incorporates the concept of inheritance which is the essence of subtype/supertype and it limits the degree of redundancy. The view approach also helps prevent the familiar 'update anomaly' problems [DAT84] which arise when the DB contains duplicates. Algorithm 1 simply ignores the possibility of this type of semantics existing in the relational model while algorithms 2 and 3 introduce steps to extract such semantics from the *user*. However, they only handle the case where such relationships are modelled by having separate base relations with common key attributes.

c). **attribute naming** - this is one of the most problematic areas in the translation process; the same attribute may be named differently in different relations (synonyms). However, a common name may be used to denote entirely different properties(homonyms). This is particularly relevant when the attributes are identifiers such as keys (primary, candidate or foreign keys). Algorithm 1 assumes unique naming of the attributes as no step is introduced to handle synonyms and homonyms. Algorithms 2 introduces a pre-processing step for renaming of attributes through querying the user. This can however, be a very tedious process. Algorithm 3, by assuming that all inclusion dependencies of the schema are given automatically rids itself of any attribute naming problems since inclusion dependencies clearly identify the synonyms. This incidentally also caters for problems with homonyms since we can simply assume that attributes in different relations are different, even if they have the same name, unless explicitly linked through inclusion dependencies. The extraction of inclusion dependencies could however, be a very tedious task requiring a lot of expertise on the part of the user.

d). **the role of candidate keys** - apart from the fact that there can be more than one candidate key for a single relation whereas only a single primary key is allowed, candidate keys can be, and often are, employed in place of primary keys. Vital schema semantics can be borne through candidate keys which could easily be ignored if the translation algorithm only used primary keys. Only Algorithms 2 and 3 consider the role of candidate keys but as will be shown later, their approach of simply swapping candidate keys with primary keys wherever they are used to denote relationdships normally represented with primary keys can lead to serious semantic difficulties and to some relationships not being shown. Such candidate keys ought to be treated independently of primary keys.

e). **the linkage between the conceptual model and the underlying relational model** - Where the conceptual model is merely used as an aid to understanding the semantics of the underlying relational model, it is important to show this connection as users still need to understand the structure of the latter before they can frame relevant queries of the DB using its query language. No algorithm to date has considered this linkage. Through this linkage, much information can be passed to the users about the semantics and the behaviour of the DB model.

f). **the extent to which the behaviour of the relational model is captured and made explicit at the conceptual level** - this criterion is cited in Davis and Arora's algorithm and is concerned with the insertion and update semantics of the conceptual model. This is a very significant contribution of Algorithm 1 which attempts to turn certain implicit constraints in the logical structure into explicit ones at the conceptual level, thus offering the designer, as the authors put it, "an ability to modify its behaviour". Their algorithm however, ignores cardinality semantics which constitute a significant part of that behaviour. Algorithms 2 and 3 introduce steps to capture cardinality constraints from the user but do not attempt to turn implicit constraints into explicit ones at the conceptual level.

3 The Entity Category relationship model (ECR)

The Entity-Relationship (ER) model is usually the favoured model for use at the conceptual level for the following reasons :-

- as a semantic data model, it supports rich semantics;
- it is easy to describe and understand the conceptual schema of a DB expressed as an ER-model.
- it allows for both structural and behavioural semantics to be described;
- it provides for easy mapping to traditional models;
- it embodies very few concepts, thus it is easy for the user to learn;
- it is widely employed for conceptual design being the basis of many CASE tools.

However, the original ER-model of Chen [CHE76] does not support abstractions of the generalisation/specialisation type and as such a variant of it supporting these additional semantics is usually adopted. One variant of this model supporting such additional semantics is the Entity-Category-Relationship (ECR) model [ELM85]. A category in the ECR model is defined as a subset of the union of one or more defining entity sets. If it is the subset of one defining entity set then it is a subclass category otherwise it is a superclass category.

Fig. 2 is a picturial representation of an Entity-Category-Relationship model. The ECR model like the ER model shows the maximum cardinality (degree) of each relationship; the one on the DEPARTMENT entity side of the Employs relationship means that an employee can belong to at most one department while the *N* on the EMPLOYEE entity side of the relationship means that a department can have any number of employees up to the maximum of N. Existence Dependency constraints are shown by enclosing the weak entity set (as it is called) in a double-rectangle on the Entity-Relationship Diagram (ERD), with an arrow pointing to this weak entity and a label of either *E* or *ID* in the associated relationship (which also becomes a weak relationship). An *E* is placed in the relationship box if the weak entity can be identified by the value (s) of its own attributes while an *ID* is used if the weak entity can only be identified by its relationship with the entity on which it is dependent. An ID dependency is automatically an existence constraint but an existence constraint is not necessarily an ID dependency. It is through existence dependencies and cardinality information that the ER and ECR models capture some of the behaviour of the DB.

4 Modelling in Relational Systems

The single modelling concept of the relational model is the *relation* which is defined as a subset of the cartesian product of its underlying domains. The relation is used to model both entities and relationships, though some relationships may also be modelled using foreign keys. The choice of whether to use a separate relation for a relationship (known as a relationship relation) or to introduce a foreign key in the related relation is usually dependent on the degree (cardinality) of the relationship; one-to-many relationships are usually supported by foreign keys while many-to-many relationships are supported through *relationship relations*. Thus to a limited extent, cardinality constraints are captured

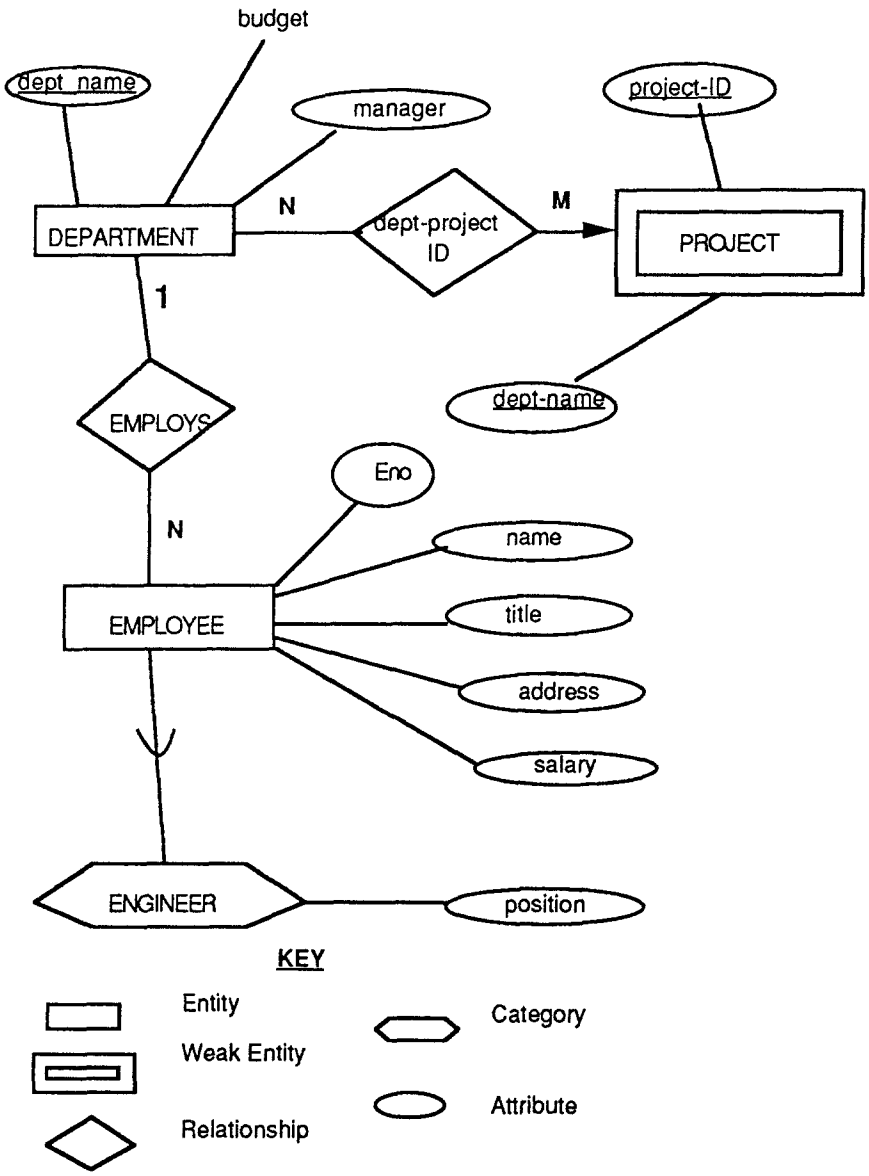

Fig. 2. An ECR Model For A Departmental DB

in the relational model. No generally agreed approach seems to exist for modelling subtypes and generalisation hierarchies in a relational model. Separate relations with common key attributes may simply be created for both the generic and subtype entities, or views could be adopted. The view approach [RAM89,RAM91] seems more intuitive since it encourages some amount of *inheritance* and also adheres to one modelling goal of the

relational model, namely that of *removing redundancies*. To ascertain that each relation describes either a single entity or a single relationship rather than multiple entities or mixtures of entities and relationships, and to remove redundancies, the relations of the relational model are usually modelled as *third normal form (3NF)* relations.

5 A New Composite Algorithm For Mapping existing Relational Schemas to ECR Models

No classification of the relations and attributes of the relational schema is needed in our algorithm. Consideration of each relation's role is automated. Like previous approaches it is assumed that the relational schema is normalised up to 3NF. We will use the sample relational schema, in Fig.3, to illustrate the different steps of this algorithm. In this schema, relation keys are shown *underlined*, candidate keys are shown in *italics* and subtype/super-type relationships are represented by using common key attributes.

person(ssn,name,address)
student(stud_id,*ssn*,sname,address)
undergrad(undergrad_id,*ssn*,year_of_study,sname,address).
course(number,name,hour)
Enrollment(cou_number,undergrad_id,date)
Employee(number,*ssn*,name,salary,building_num,room)
Employee_project(empnum,proj_num,hours_spent)
Department_project(deptnum,projnum,buget).
Job(job#,description,salary_range)
Employee_job(empnum,jobnum)
Location(building#,room,Description,Capacity)

Fig. 3. Sample Relational Schema

Steps of the Algorithm

Step 1. The Preprocessing Step - Renaming of attributes - identifying attributes such as keys and candidate keys must be uniquely named throughout the relational schema. To achieve this uniqueness of naming all synonyms must be identified and changed to a common name while all homonyms are given new names. Attribute names carry some limited semantics which however, is at too high a level to be interpretted by a computer. Knowledge of synonyms and homonyms must be supplied by the user. Relation names are used to achieve uniqueness in attribute naming throughout the schema by prefixing attribute names with relation names. If a relation does not exist for the particular attributes in question, as for relations only sharing common parts of their key attributes (not the whole key), then the user must be queried to supply a name; for example, the relations Employee_project and Department_project of Fig.3, share a common attribute represented by the synonyms *proj_num* in Employee_project and *projnum* in Department_project. After this step the schema of Fig. 3 is converted to Fig. 3.1 below.

person(ssn,name,address)
student(student_stud_id,*ssn*,sname,address)
undergrad(student_stud_id,*ssn*,year_of_study,sname,address).
course(course_number,name,hour)
Enrollment(course_number,student_stud_id,date)
Employee(employee_number,*ssn*,name,address,salary,
location_building#,location_room)
Employee_project(employee_number,project#,hours_spent)
Department_project(dept#,project#,buget).
Job(job#,description,salary_range)
Employee_job(employee_number,job#)
Location(location_building#,location_room,Description,Capacity)

Fig. 3.1. Sample Relational Schema After Step 1.

Step 2. Subtype/Supertype relationship establishment.
This step is necessary at this early stage of the algorithm so that inferrable relationships need not be computed and represented. Where views are employed to capture semantics of the subtype/supertype nature, the establishment of the subtype/supertype relationship is trivial as this is implicit in the view definitions. An example will make the process more apparent. Consider a portion of a DB schema described by the following Prolog facts:

R1 : student(stud_id,sname,address)
R2 : undergrad(undergrad_id,year_of_study)
V : undergrad_view([[undergrad_id,undergrad],[year_of_study,undergrad]
[sname,student],[address,student]],[[undergrad_id,stud_id]])

Where the *R s* stand for relation schemes and the*V* represents a view with each attribute of the view shown together with the relation from which it is derived and the *'where'* part of the view definition shown as the second argument. We assume suitable routines exist to extract the view definitions for subtype to supertype relationships and present them in the above format. Our algorithm simply stores information about the subtype/supertype relationship from R2 to R1. The process of identifying subtype/supertype relationships is more demanding when these relationships are simply captured through common keys and duplicated attributes. In this case, any two or more relations having the same key attribute(s) or where the primary key of one matches the candidate key of the other are presented to the user for confirmation of any subtypes/supertype relationships. Those attributes of the subtype relation that can be inherited from the supertype are deleted. During this step the relations Person, Student, Undergrad and Employee of Fig. 3.1 will be isolated and presented to the user. These relations will be modified as described above and subtype/supertype information stored as shown in Fig 3.2. Notice that candidate key attributes are maintained in the subtype entity since the relationship can be represented using candidate keys.

person(ssn,name,address)
student(student_stud_id,*ssn*)
undergrad(student_stud_id,*ssn*,year_of_study)
Employee(employee_number,*ssn*,salary,location_building#,
location_room)
subtype(person,student)
subtype(person,employee)
subtype(student,undergrad)

Fig. 3.2. Schema after Step 2 showing subtype/ supertype relationships and modified relations

Step 3 Isolation of regular entities

Algorithm 1 gives two rules for isolating regular entities out of relations of the schema. No user interaction is needed in this case. The rules are:

a). relations with a single attribute as their primary key are converted into entities.

b). relations with more than one attribute making up their key are examined and if this key is always used as a whole in other relations, and never used as disjoint parts separately in the keys of other relations, or if its attributes are never used again (i.e. as a whole or partially) then the corresponding relations are also converted to entities. During this step, the relations Person, Student, Undergrad, Course, Employee, Job and Location are converted into entities having the same attributes and keys as the converted relation.

In the following steps of the Algorithm we make reference to relation ID to stand for either the primary key of a relation or its candidate key. Relationships between entities can be represented using primary keys or candidate keys. Apart from the subtype/supertype relationship already described above, three other cases exist when candidate keys could play the normal role of a primary key :

i) the candidate key of some relation R1 can occur as part of the primary key of another relation R2, where R2 is either a weak entity (see Step 4 below) or a relationship relation (step 5).

ii) the candidate key of a relation, if formed by concatenation of primary keys of other relations, would denote a relationship relation between these entities.

iii) the candidate key of a relation can also occur as a non-key attribute of another relation to denote the one-to-many relationship between the former entity relation and the latter.

Algorithms 2 and 3 while ignoring case iii) above suggest that for case i) above, the primary key of R1 replaces its candidate key in R2 while for case ii) the candidate key should become the new primary key of its relationship. The general problem with both suggestions is that the resulting conceptual model could mislead the user into making incorrect logical level joins. The specific problem with the second suggestion is that making the candidate key the new primary key can lead to certain relationships, represented through the old primary key not being found by the algorithm. For example, consider the sample of part of a typical relational schema given in Fig.4.

person(ssn,name).
employee(ssn,salary).
student(collegeno,*ssn*).
course(courseno,coursename).
enrollment(collegeno,courseno,date)

Fig 4. Part of a typical relational schema

Following the suggestion of algorithms 2 and 3, the primary key of student becomes *'ssn'* since a subtype/supertype relationship would naturally be confirmed by the user between the *student* and *person* entity relations. This change will mean that the relationship relation *'enrollment'* will be wrongly treated as a weak entity dependent on the course entity rather than a many-to-many relationship between students and courses. In our algorithm, candidate keys are considered independently but in much the same way as keys for possible hidden relationships. This means that in the ECRD diagram, the relationship key is not restricted to being a concatenation of the Primary keys of its entities but rather to being a concatenation of identifiers of its participating entities. We will consequently use the name ID to refer to either the Primary key or the Candidate key of a relation.

Step 4 Isolation of Weak Entities

This step also derives from algorithm 1 but with some modification to take the subtype/ supertype relationships created so far and candidate keys into consideration. The approach is to compare the ID of the remaining relations of the relational schema with the ID of the entities derived so far. If one or more attributes of a non-relationship relation ID are left over, then this relation becomes a weak entity. The attribute(s) left over when the relation ID is compared to the entity ID is known as a *dangling key* and forms the only attribute(s) of the weak entity. A relationship is created between the new weak entity and the regular entity. The key of this relationship is made up of all the attributes of the original relation from which the weak entity derives. Where the entity used for comparison is involved in a generalisation hierarchy, then the user must be queried to see at which level of the hierarchy the relationship should be introduced and any other entities in the hierarchy could be ignored in this comparison. Information about the type of dependence of the weak entity on the regular entity as well as on the cardinality of the relationship has to be extracted from the user. In Algorithm 1, the cardinality of the relationship is simply made to be many-to-many. We argue that this may be too general to convey the specific semantics of the relationship.

From the remaining relations after Step 3 above, the Employee-project relation becomes a weak entity since the key of the Employee entity-set is employee_number and the key of the Employee_poject relation is (employee_number,project#). The project# is the dangling key attribute. A new entity-set Project will be created (after consulation with the user for its name) with its only attribute also forming its key. A relationship is created between the new weak entity and the entity on which it is dependent. The new Project entity-set will give rise to another weak entity from the Department_project relation, named Department with its only attributes being the dangling key (dept#,budget). This weak entity will be treated in like manner to the previous weak entity-set.

Step 5: The Many-to-Many Relationships
As explained in Section 4 above, many-to-many relations are normally modelled in relational systems by creating new relations and concatenating the IDs of the involved entities to form the ID of the new relation. Thus in the reverse process, any relation whose ID is made up of a concatenation of the IDs of entities that have been derived so far is turned into a many-to-many relationship between the involved entities. Again subtype/supertype relationships of the entities must be taken into consideration and the user queried if necessary to see at which level of the hierarchy the relationship should be introduced. The Employee_job relation and the Enrollment relation meet these criteria. Thus a many-to-many relationship will be created between their participant entities. In the case of the Enrollment relation, the user will need to say whether the relationship is between the Entity-sets Student and Course or Undergrad and Course. Owing to the possibility of a many-to-one relationship (see step 6) being modelled using a relationship relation, the user would have to be queried as to the exact cardinality of the relationship; i.e. whether the cardinality is many-to-many or many-to-one.

Step 6: The Many-to-one Relationships
By this stage, all the relations of the relational schema would have either been converted into regular entities, weak entities and their relationships, or many-to-many relationships. Many-to-One relationships are usually modelled in the relational model by including the ID of the entity on the One side of the relationship in the set of attributes of the entity on the Many side of the relationship. Thus if the ID of an entity that has been derived occurs as a non-key attribute in another entity, then a one-to-many relationship is created from the former entity to the latter. The exact cardinality of the relationship (i.e. one-to-one or one-to-many) can be found by querying the user. Entities involved in a generalisation hierarchy must be treated in like manner to the previous two steps. The user must also be queried to provide a suitable name for the relationship as well as any possible existence dependencies between the entities. A many-to-one relationship will be created, using this step, between the entity-sets, Employee and Location. This step completes the algorithm. The ECR Diagram derived in this way from the relational schema of Fig 3 is shown in Fig. 5.

6 Implementation of the Algorithm

All 6 steps of the algorithm described above have been implemented in the Arity/Prolog implementation of Prolog on an IBM PC machine [MAR86,ARI87]. This implementation of Prolog extends the standard Prolog version with very useful extralogical features to improve its performance and to cater for more algorithmic tasks. Amongst these features are *virtual memory management* capabilities and *text screen management* functions useful for implementing graphics. The graphics function was very useful in implementing the Entity-Category-Relationship Diagram (ECRD) of the conceptual model.

Step 1 of the algorithm described above requires that knowledge of synonyms and homonyms be supplied by the *user*. To facilitate this process, the system displays pairs of

relations from the relational schema at a time and then asks the user to identify any possible synonyms and homonyms. As relation names are unique within a single database schema, uniqueness in attribute names is achieved by prefixing the individual attributes with the name of the parent relation. This approach appears more practical and easier to apply than previous ones which either require unique naming (Algorithm 1) or that all inclusion dependencies be stated and the relations and attributes classified. Algorithm 2, while recommending such renaming does not show how this can be achieved.

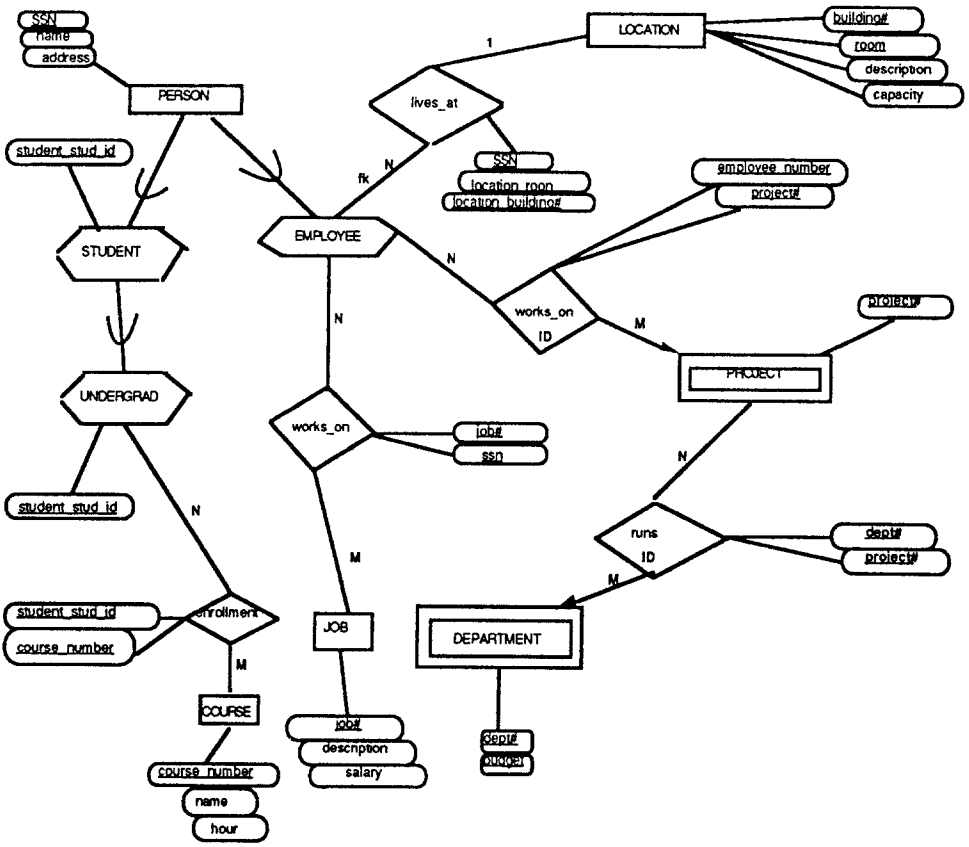

Fig. 5. An Entity-Category Relationship Diagram of the Sample Relational Schema.

Whenever input is required to test if a certain rule can 'fire', the system gives the user the choice of asking 'why' such input is needed; in which case the system would display the rule for which input is sought and then repeat the question. The rules have been structured in an easy-to-read manner and should assist the user gain a better understanding of the algorithm's reasoning process as well as the semantics of the DB. Example rules of this nature that may require input from the user include :

- rules that capture cardinality constraints of relationships;
- rules for capturing dependency constraints of weak entities;

- rules for determining at what level of a generalisation hierarchy a relationship should be introduced.
- rules that capture new relationship names.

The last set of rules involve relationships represented by foreign keys (i.e. keys, or more appropriately IDs, of some relations occuring as non-keys of other relations). Queries for rules, such as the last set, that capture relationship names, normally present the user with some conclusion that the system has drawn, for example, that a many-to-one relationship exists between some two entities. The *user* is given the option in this case to ask for an explanation of how this conclusion was arrived at; in which case the domain knowledge embodied in the rule for establishing many-to-one relationships from foreign keys is shown. The condition parts of this rule would have been instantiated. The system thus, exhibits some expert system capabilities. However, as some steps of the algorithm are fully automated and do not require input from the user, complete knowledge of how all steps of the algorithm are carried out cannot and need not be shown to the *user*.

A sub-module of our system is concerned with the graphical output. An Entity- Category Relationship Diagram (ECRD) of the model is automatically generated from the structures generated by the various steps, of the algorithm. These structures represent entities, relationships and relationship constraints. Some labels have been introduced in this ECRD to show the user how certain relationships of the conceptual model are represented at the logical level, thus assisting the user to make extensional database queries from a knowledge of this conceptual model. Where the relationship is represented by a foreign key in the relational schema, the label *"fk"* is placed on the line connecting the relationship diamond and the entity rectangle (or hexagon) in which the foreign key is placed; otherwise it is to be understood that the relationship and its participating entities are modelled at the logical level in the same way as at the conceptual level, i.e. by separate relations with the same attributes and the same keys. For subtype/supertypes captured at the logical level through views, a label *V* is placed on the line connecting the supertype entity to its subtype.

7 Conclusions

In this paper, we examined three existing representative algorithms for converting relational schemas to conceptual models. Through this comparative study, we identified the important contributions of each algorithm as well as its limitations. It became apparent that some of the limitations of one algorithm were addressed by another algorithm while some still remained to be solved. Thus a new algorithm was developed that integrated the important concepts of these algorithms and introduced new concepts to rectify their limitations. This new algorithm is more general and easier to apply than any of its predecessors. Its implementation also shows a high degree of automation. While certain steps of the algorithm are made transparent to the user in the implementation, users are given the choice to see some of the rules that implement other steps in an attempt to give them a better understanding of that part of the algorithm and as such make it easier for them to provide the algorithm with the correct input. Thus our approach adopts some

Expert System techniques useful in explaining parts of the algorithm to the user employing it. This should make the algorithm easier to use and available to a wider group of DB users.

Due to designer decisions, the exact approach used for modelling particular relationships at the logical level can be different. At the conceptual level however, relationships are modelled in the same fashion. Thus we found it necessary to augment the conceptual model represented by the ERD with new features (labels on arcs) that show how such relationships are modelled at the logical level. These new features should make the correspondence between the conceptual model and the underlying logical model more apparent. Thus, users querying the data of the database are assisted by the conceptual model which helps them see relationships and how they are modelled in the relational model.

One major contribution of our algorithm is in its treatment of subtype/supertype relationships and candidate keys. Subtype/supertype relationships are created at an early stage in the algorithm and any generalisation hierarchies built and maintained for use in later stages. As the entities in a subtype/supertype hierarchy have common IDs, when this ID is found as an attribute of some other relation(s) (i.e. as a foreign key), the system first presents the most generic entity in the hierarchy together with the other entity (the one with the foreign key) and asks the user if the relationship exists. If the response to this is 'yes', then no further entities in the hierarchy will be considered. If the answer is 'no', then the entities at the next level of the hierarchy will be considered. In this way redundant relationships are not be created. By treating candidate keys in the same way as primary keys when creating the relationships of the conceptual model, it became possible to identify the relationships embodied by these candidate keys without compromising those embodied by primary keys. This leads to a semantically richer conceptual model and one from which the logical model can be easily understood by its users; which should lead to correct utilisation of its semantics.

In [KAL91], an implementation followed by a critique of Algorithm 2 was carried out. One criticism pointed out in this work is that the algorithm cannot handle multi-level dependencies properly, e.g. if a weak entity A depends on another weak entity B which depends on a regular entity C, then the algorithm does not show the dependency of A on B since an entity can only be ID dependent on a regular entity. As pointed out in Step 4 of our algorithm, multilevel dependencies are automatically catered for. However, in the current implementation the concept of inheritance is not used; thus for the example above, A and B will be shown dependent on C as well as A being dependent on B. The fact that A is dependent on C is implicit in its being dependent on B which in turn is dependent on C. The modification needed for this is however, not difficult to implement. A's dependency on C could simply be deleted.

A useful future extension of this system would be to investigate how the semantics of the DB that are coded in its application programs can be used to aid the mapping process. The semantics captured in application programs are typically of two types; namely state constraints, e.g. to enforce a subrange constraint on an attribute of a relation defined in the

schema as one of the basic types of the DBMS, and transition (behavioural) constraints which monitor changes made on the database. Another extension would be to introduce steps that can be used to first derive a global schema for a Multidatabase System(MBS) in relational form, and then employ the algorithm described in this paper to produce a global conceptual schema of the entire information of interest to the user.

A fundamental limitation of the whole approach to eliciting the semantics of a relational database as presented in this paper and previous similar works, is a general lack of consideration of the explicit constraints of the database, examples of which are cardinality constraints and dependency constraints. Explicit constraints are those constraints that are tangential to the data model and serve to augment the structure specification of the database. Constraints captured with the structure of the data model are called inherent constraints. The relational model is well known for being weak on capturing inherent constraints and as such a great deal of the semantics of the database is captured through the explicit constraints. No known diagrammatic approach exists for completely capturing all types of these explicit constraints. Thus, some other way must be found for making their semantics apparent to users. Some of these constraints actually refer to particular instances of the database. A simple approach may be to simply output the logic specification of these constraints together with the conceptual model and leave it to the user to interpret the output. The problem with this is that a significant part of the logical schema (up to 80 percent) could be made up of the explicit constraints, [DAT84].

A further extension of our work is to investigate how these explicit constraints can be employed in the Multidatabase environment, (where most difficulties arise regarding the semantics of a database; usually the semantics of remote databases), to enhance the user's knowledge of the semantics of these databases. These explicit constraints could be used to provide intensional answers to certain data related queries whenever the system, from knowledge of the explicit constraints, deems that such semantics need to be made clear to the user. This idea borrows from the recent research interest in intensional query processing in deductive databases [CHO87,IMI87,PIR30,SON90]. Thus a complete conceptual model of a database must embody its inherent and explicit constraints as well as other structural aspects.

Reference

[ABR74] J.R,Abrial, *"Data Semantics"*, in Data Base Management Etd by J.W. Klimbie & K.L Koffeman, North-Holland, pp1-59, 1974

[ALL82] F.W.Allen,M.E.S.Loomis & M.V.Mannino, *"The Integrated Dictionary/ Directory System"*, - Computing Surveys 1982.

[ARI87] Arity Corporation, *"Arity/Prolog Programming Manual"* - Arity Corporation - 1987.

[BAT86] C.Batini et al; *"A comparative analysis of database schema integration"*, ACM comp. surveys, Dec. 1986.

[BRA76] G. Bracchi, P. Paolini & G. Pelagatti, *"Binary Logical Associations in Data Modelling"*, in Modelling in Data Base Management Systems editted by G.M.

Nijssen, pp. 125-148, North Holland, 1976

[CER84] S.Ceri & G.Pelagatti, *"Distributed Databases : Principles and systems"*, McGraw-Hill Int., 1984.

[CHE76] P.P. Chen, *"The Entity Relationship Model -- Towards a Unified View of Data"*, ACM TODS Vol. 1, No. 1, pp 9-36, March 1976.

[CHO87] L. Cholvy & R. Demolombe,*"Querying a Rule Base"*. In Procs. of the 1st International Conf. on Expert Database Systems, Pp365-371, S. Carolina - 1987. L. Kerschberg (Etd).

[COD70] E.F. Codd, *"A relational Model of data for large shared databanks"*, Communications of the ACM 1970.

[COD79] E.F.Codd, *"Extending the Database Relational Model to Capture more meaning"* ACM Trans. on Database Systems - 1979.

[DAT84] C.J. Date, *"Database Systems"*, Vols.1 & 2 , (4th Edition) - Addison-Wesley Pub -1984.

[DAT87] C.J.Date *"What is a Distributed Database System?"* , InfoDB, Vol 2, No 2,1987.

[DAV88] K.H. Davis & A.K. Arora, *"Converting a Relational Database Model into an Entity-Relationship Model"*, Seventh International Conference on Entity-relationship Approach", 1988.

[DOL87] D.R.Dolk & R.A.Kirsch, *"A Relational Information Resource Dictionary System"*, Comm. of the ACM - 1987.

[ELM85] R.Elmasri, J.Weeldreyer & A.Hevner, *"The Category Concept : An Extension to the Entity-Relationship model"*, In Data and Knowledge Engineering, Vol. 1, No. 1, June, 1985.

[ELM89] R.Elmasri & S.B.Navathe, *"Fundamentals of Database Systems"*, Addison-Wesley Pub. - 1989.

[GAR85] G.Gardarin & M.Jarke; *"Database Design Tools: An Expert Systems Approach"*; VLDB 1985.

[GAR90]. G.Gardarin & P. Valduriez, *"Relational Databases and Knowledge Bases"*, Adison-Wesley Pub-Co -1990.

[GRA88] P.M.D. Gray,G.E.Storrs & J.B.H. du Boulay, *"Knowledge Representations for database meta-data"*, - AI review 2,3-29 1988.

[GOT90] G.Gottlob, L.Tancia & S.Ceri,*"Surveys in Computer Science"*,Springer -Verlag - 1990.

[HAM81] M.Hammer & D.McLeod, *"Database Description with the SDM"*, ACM Trans. on DBSs, Sept 1981.

[IMI87] L. Imielinski, *"Intelligent Query Answering in Rule Based Systems"*, Journal of Logic Programming, 4(3):229-258, Sept, 1987.

[JAR84] M.Jarke & Y.Vassiliou, *"Databases and Expert systems: Opportunitiesand architectures for integration"*, - from New applications of DBs Eds. G.Gardarin & E. Gelembe - 1984.

[JOH89] P.Johannesson & K.Kalman, *"A Method for Translating Relational Schemas Into Conceptual Schemas"*, 8th International Conference On Entity-Relationship Approach, 1989.

[JOH84] R,G.Johnson; *"Integrating Data and Meta-data to enhance the user inteface"*,

BNCOD-4 1984.

[KAL91] K.Kalman, *"Implementation and Critique of an algorithm which maps a Relational Database to a Conceptual Model"*, CAiSE 1991.

[KEN79] W.Kent, *"Limitations of Record-Based Information Models"*, ACM TODS 1979.

[LIT88] W.Litwin, *"From Database Systems to Multidatabase Systems:Why and How"*; BNCOD-6 1988.

[MAR86] C. Marcus, "Prolog Programming- Applications for DBSs,Expert Systems and Natural Language Systems",- Arity Corporation - 1986.

[NAV88] S.B. Navathe & A.M. Awong, *"Abstracting Relational and /Hierarchical DataWith a Semantic Data Model"*, 7th International Conference on Entity-Relationship Approach- 1988.

[OMO89] A.O.Omololu,N.J.Fiddian & W.A.Gray, *"Confederated Database Management Systems"* - 7th British National Conference on Databases Etd. by M.H.Williams 1989.

[OXO87] E. Oxborrow, *"Databases and Database Systems"* - 1987

[PEC88] J. Peckham & F. Maryanski, *"Semantic Data Models"*, - ACM Computing Surveys,Vol.20,No.3,Sept.,1988.

[PIR89] A. Pirotte & D.Roelants, *"Constraints for Improving the Generation of Intensional Answers in a Deductive Answer"*, In the Procs. of the International Conf. on Data Engineering, Los Angeles, California - 1989.

[RAM89] A. Ramfos,N.J.Fiddian & W.A.Gray, *"Object-Oriented to Relational Inter-Schema Meta-Translation"*, 1989 WorkshopOn Heterogeneous Databases, Dec. 11- 13, 1989, Chicago.

[RAM91] A. Ramfos,N.J.Fiddian & W.A.Gray, *"Relational to Object-Oriented Schema Meta-Translation"*, Procs. of the 9th British National Conference On the management of Data, Wolverhampton 1991.

[ROU87] N.Roussopoulos & L.Mark *"Information Interchange between self-describing Databases"*, IEEE Trans. on Data Engineering 1987.

[SMI77] J.M.Smith & D.C.P.Smith, *"Database Abstractions: Aggregation and Generalisation"*, ACM TODS-1977

[SMI81] J.M Smith et al; *"Multibase - Integrating Heterogeneous Distributed Databases"*, Proc. AFIPS Nat. Comp. Conf. 1981.

[SON90] I-Y.Song,H-J.Kim & P. Geutner, *"Intensional Query Processing: A Three-Step Approach"*, Procs. 1st International Conf. on Databases and Expert Systems Applications, Vienna 1990.

[TSI82] D.C Tsichritzis & F.H. Lochovsky, *"Data Models"*, Published by Prentice Hall Inc. - New Jersey - 1982.

[VAS83] Y.Vassiliou & M.Jarke; *"How does an Expert Syustem get its data?"*, VLDB, 1983.

[VAS84] Y. Vassiliou & M. Jarke; *"Databases and Expert Systems: Opportunities and Architectures"*, New applications of Expert Systems and Databases, Edited by G.Gardarin, 1984.