The Problem of Identification

Ulrich Schiel *

Universidade Federal da Paraíba
Dep. de Sistemas e Computaçao
Av. Aprígio Veloso, s/n
58100 Campina Grande/PB
BRAZIL

The existing models for database applications use a few fundamental concepts for the description of the world to be modeled. The most common of these concepts is the ***entity***, also called object, element, concept, information, token, data or value. Entities are identifiable and distinguishable existing things of the real world. In general the concept of entity is used only for static things whose existence does not depend directly from other entities, in contrast to the dynamic happenings called ***events*** and the existence of some dependency between two or more entities called ***relationship*** (or attribute or property) which can exist only in conjunction with the related entities. More complex combinations of entities, relationships and events we call ***facts***. Therefore "*John has broken his leg yesterday and is in Santa Clara Hospital*" is a fact which combines the event of "*breaking a leg*" occured with the entity "*leg of John*" at time-event "*yesterday*" and the relationship "*is in*" valid between "*John*" and "*Santa Clara Hospital*" now. We use the term object to refer to entities, relationships, events or facts, undistinguishably.

Entities are considered to be of a certain ***type*** or belong to a ***class***. This class must not be unique. We denote this relation by ":". For instance, e:EMPLOYEE, e:PERSON, n:AGE, n:INTEGER. In order to distinguish entities from its identification we use parethesis, i.e. (k) means the entity identified by "k". Between classes some special hierarchical relationships (predicates) may hold, whose validity extend to the elements. If C and D are classes, the following predicates are applicable:

*actually at GMD-IPSI, Dolivostrasse 15, 6100 Darmstadt, FRG
ean: schiel@ipsi.darmstadt.gmd.dbp.de

is-a(C,D) : each element of C is an element of D;
part-of(C,D) : elements of D are aggregates and have one component with elements of C;
elem-of(C,D) : elements of D are sets of elements of C.

These are the three well known abstractions called generalization, aggregation and grouping [SFNC-84].

After having introduced briefly this general concepts, we go on to analyse the question about the identification of objects, in particular the entities.

In order to process objects by an information system, they must be ***represented*** in the computer and the results of the computation must be correctly ***interpreted***, formated and printed out. The way of an object from the outside real world to its internal representation in a computer file can pass through several representation levels on paper, terminal, main memory, and so on. The problem of representation and interpretation of objects has not received much attention in the past, because two solutions were sufficient for the most conventional systems:

- for entities like strings and numbers a uniform codification for each level was defined;
- for more complex entities a predefined key identifies uniquely the entity in the environment.

For more advanced applications these two categories aro no longer sufficient. User-friendly interfaces, AI applications and distributed systems calls for concepts like weak- or context-dependent identification, identification patterns and mispelling correction.

For each representation level the solution for a good identification may be quite different, the external level need a natural identification and the internal levels need technical optimal solutions. But mainly with the development of Artificial Intelligence software and hardware the frontiers of these levels become fuzzy and we will treat the question in a uniform way.

Searle [Se-71] states that an object can be identified in two ways: giving them a proper name or describing it. The first way has several advantages: the object has a unique, short and constant identification. But in several cases entities has no natural names and a code must be created. In other cases, the existing (natural) name is not completely unique or we know only some properties of the entity and want to retrieve the entity by these properties. In this case it must also be considered that the properties of an object can change over time.

The most radical solution for the identification of entities is the use of ***surrogates*** [Co-79, So-84]. These are completely abstract codes which identify uniquely an entity in the whole database. Thus, a person '*John*' and a salary of 2,000.00 are given as *John*-#10340 and 2000.00-#10352, where the symbol # caracterizes surrogates. They may be useful for the

internal processing and representation of complex entities which appear in several contexts of the database, in order to represent relationships, facts or aggregated entities. For the internal representation of elementary values like strings and numbers a bether solution is using the internal bit-code of the numbers and strings (as surrogate). For the external level surrogates and in some cases also user-defined keys are useless.

Let us suppose, for instance, an information system about thousands of persons (Insurance Co., Phone service or other) with frequent queries and updates. It is certainly not the best solution to require each customer to have its insurance number at hand in order to obtain some information or make an update. In this case, the most natural identification is the proper name and must be maintained with this characteristic. Even if in some cases duplications occur (homonyms), it is the best identification. Therefore it will be called a ***default identification***. If a duplication occurs, additional secondary identifications like birthday or parents, must be used.

The example below shows that in several cases natural identification is possible but, if it is not unique, additional measures must be taken in order to

- reduce the context or universe stepwise up to that the identification turns to be unique, i.e. the default identification becomes an absolute identification, or
- accept the duplicates in which case the user must make a manual selection.

Another liberalization of strong identification requirenments is the so-called ***unresolved reference*** [RF-88]. This means that if we have a field reserved to contain a reference to an object (a pointer, key or address), but for some objects we do not have this information at hand, then we store, preliminarly, some other information about the object (e.g. a string with its name) and later on the reference may be resolved with the definite identification.

The one problem is to determine the correct identification for a new entity, the other is, having a key, determine whether the system can retrieve the correct entity. Let us illustrate this problem with an example. Suppose we have a key '124', what is the meaning of '124'? It can be, among others, one of the following [Sc-84]:

124 - the number one hundred and twenty four;
(124) - an entity identified by the number 124;
(124:REG-NUMBER) - the entity identified by the register number 124;
(124):EMPLOYEE - the employee identified by the number 124;
(124:REG-NUMBER):EMPLOYEE - the employee identified by the register number 124.

If we change 124 by #124 and the word *number* by *surrogate* (or pointer) we obtain another set of cases. Note that only the first and the last cases are completely context-free. The other need the context of a program or statement and some previous type or parameter definitions.

Resuming the questions discussed above, we give a classification of entity identification forms:

1) **Self-identifying entities**: these are primitive entities so closely related to their own representation (on paper or on the computer) that no distinction is made between the thing and its representation. Examples: 2,000.00:DOLLAR, 'Jose da Silva':STRING, 124:INTEGER.

2) **Surrogates**: complex entities without natural name and which appear in several contexts. Internal adresses (pointers) can be considered a special case of surrogates. Example: (#2001).

3) **Keys**: entities without natural name but for which was defined a well known code. Examples: (109981049:INSURANCE-NUMBER), (01108:REGISTER-NUMBER):SOCIETY-MEMBER.

4) **Proper names**: entities having a natural name which identifies them completely (absolute identification) or in some context defined by some additional properties (default identification). Example: ('Jose da Silva`):SOCIETY-MEMBER or ('Jose da Silva`:NAME, 06-06-46:BIRTHDATE):SOCIETY-MEMBER. This case also applies to dynamic objects like procedures and events, by calling their names.

5) **Description**: entities without explicit identification. They are retrieved by enumerating some properties they must satisfy. In an SQL database the only way to retrieve an entity (tuple) is to state these properties in the WHERE-clause. This case also applies well to more complex facts which are not explicitly in the database, but can be obtained combining relationships and entities. Example: ('blue`:EYES, 'black`:HAIR, n:AGE, n < 24) :PERSON. In the case of text files or long fields, the text contains information about several objects which are identifieable by giving retrieval patterns.

6) **Non-identified**: objects not existing as permanent individuals in the system. If, for instance, one information of our system is the existence of 1000 nails, each of these nails is an object, but we are not interested to create 1000 identifications for them. Another case occurs when we execute an online query on the terminal. The query itself is a dynamic object and all the intermediate results of processing the query (e.g. in relational algebra) have no visible identification. A third case of this category are not explicitly stored intensional objects of deductive databases.

Note that at distinct representation levels the same entity can obtain different kinds of identification. In the case of unresolved references this also occur at the same level.

Relationships are generally represented as tuples of the related entities. Therefore they inherit the problems of entity identification. In case of the abstractions generalization, aggregation and grouping the following considerations are necessary.

For a *generalization* "is-a(C,G)" the identification of the elements of G can also be used for C because C inherits all properties of G. But in several cases C has a proper identification generally stronger than that of G. For instance, in case "is-a(EMPLOYEE, PERSON)" persons can be identified by names and employees may have an additional key, its register number. In this case those elements of PERSON which are employees should be identified by its register number too. For more complex generalization trees, Schrefl and Neuhold [SN-88] suggest to introduce an ***object colouring*** in order to re-identify the same entity in several classes. This is given by the "*as*" connector. Suppose the additional generalization "is-a(FEMALE, PERSON)" then (12311:REG-NUMBER)*as*FEMALE means the identification of the employee (12311) as member of the class FEMALE.

In an *aggregation* each aggregated entity is a composite of its parts $a=<p_1,p_2,...,p_n>$ and therefore if p_i, i=1,..,n are the identifications of the components, the tuple $<p_1,...,p_n>$ is an identification of the aggregate. But the aggregate can have a proper key a=(k) and we must be careful that an alteration of one component p_i to p_i' arises to another identification what means another aggregated entity. This is the main difference between defining a class A as an aggregation of classes $C_1,...,C_n$ or defining n relationships between A and C_i, i=1,...,n. the changing of one or more of the n relationships does not change the entity itself. Suppose as a critical example a car and its motor. Is the motor a component of the car, i.e. does "part-of(MOTOR,CAR)" hold or does exist a relationship 'has-motor' from CAR to MOTOR? If we replace the motor by another, is the result the same car? Herewith we come to the philosophical problem of object identity. Suppose the fact *'she was never the same, after here dog died'.* Does this mean that she is another entity after here terrible experience, and therefore need another identification ?

For the *grouping* abstraction, suppose a class of EMPLOYEES with a relationship 'has-leader' from EMPLOYEES to EMPLOYEES. Another class called TEAMS contains groups of employees having the same leader, a room and a budget. A natural identification of the team is via the leader, but formally 'has-leader' is a relationship between EMPLOYEES and must be inherited upwards to TEAM in order to identify the groups.

The last consideration about identification is that an "intelligent" system must also be able to accept some mispellings in the digitation of an identification. For those cases a function must search for the nearest candidate to be a correct key. Bickel [Bi-87] suggests that the measure can be given by the name with the greatest number of (weighted) coincident letters. In our opinion this works well for proper names, but for other names, keywords, mnemonics and other, the proximity function

must take into account the relative position of each caracter, transpositions, typical variants and abbreviations.

Our intention in this short paper was to uncover some details concerning identification of objects which come across more during implementation and maintenance than during the design of information systems.

ACKNOWLEDGEMENTS

The author is grateful to Dietrich Fischer of GMD/IPSI who reads a draft of this paper and makes several helpful comments.

REFERENCES

[Bi-87] M.A. Bickel, "Automatic Correction to Mispelled Names: A Fourth-Generation Approach" *CACM vol.30*, no. 3, 1987.

[Co-79] E.F. Codd, "Extending the Database Relational Model to Capture more Meaning", *ACM-TODS vol. 4*, no. 4, 1979.

[RF-88] L. Rostek & D. Fischer, "Objektorientierte Modellierung eines Thesaurus auf der Basis eines Frame-Systems mit graphischer Benutzer-Schnittstelle", GMD-IPSI, Nachrichten für Dokumentation, Aug. 1988.

[Sc-84] U. Schiel, "A Semantic Data Model for Conceptual Schemata and its Mapping to Internal Relational Schemata", Dr. Dissertation, Univ. Stuttgart, 1984.

[SFNC-84] U. Schiel, A.L. Furtado, E.J. Neuhold and M.A. Casanova, "Towards Multilevel and Modular Conceptual Schema Specifications", *Inform. Systems vol. 9*, no. 1, 1984.

[Se-71] J.R. Searle, "The Problem of Proper Names", in *Semantics*, D. Steinberg and L. Jakobovits (eds.), Cambridge, 1971.

[So-84] J.W. Sowa, *Conceptual Structures*, Addison-Wesley, 1984.

[SN-88] M. Schrefl & E. Neuhold, "Object Class Definition by Generalization using upward Inheritance", Proc. of the 4th Intl. Conf. on Data Engineering, Los Angeles, 1988.