DATA ABSTRACTIONS: WHY and How?*

Robert C. Goldstein  
COMMERCe AND BUSINESS ADMINISTRATION  
UNIVERSITY OF BRITISH COLUMBIA  

Veda C. Storey  
DEPARTMENT OF COMPUTER INFORMATION SYSTEMS  
GEORGIA STATE UNIVERSITY  

September 1, 1998  
GSU CIS Working Paper  
CIS-98-2
DATA ABSTRACTIONS: Why and How?

Robert C. Goldstein
Faculty of Commerce and Business Administration
University of British Columbia
2053 Main Mall
Vancouver, B.C.
Canada V6T 1Z2
Email: bob.goldstein@ubc.ca

Veda C. Storey
Department of Computer Information Systems
College of Business Administration
Georgia State University
Atlanta, Georgia 30302-4015
E-mail: vstorey@gsu.edu
Phone: 404-651-3894; Fax: 404-651-3842

September 1, 1998

*This research was supported by the University of British Columbia and Georgia State University. The authors wish to thank Blaize Horner Reich, Yair Wand, and Heng-Li Yang for their helpful comments on an earlier version of this paper. The assistance of Victor Ng in implementing and testing the Oracle implementations is especially appreciated. ©1998 Robert Goldstein and Veda Storey. All rights reserved. Draft only. Do not quote or circulate.
DATA ABSTRACTIONS: Why and How?

Abstract

The relational data model has become the standard for mainstream database processing despite its well-known weakness in the area of representing application semantics. The research community's response to this situation has been the development of a collection of semantic data models that allow more of the meaning of information to be represented in a database. The primary tool for accomplishing this has been the use of various data abstractions, most commonly: inclusion, aggregation, and association. This paper develops a general model for analyzing data abstractions, and then applies it to these three best-known abstractions.

Keywords: Data abstractions, inclusion, aggregation, association, relational data model

1 INTRODUCTION

The relational data model has become the de facto standard for most real-world database management. It has the very strong appeal of a single representation structure that is, superficially at least, very similar to the two dimensional tables that humans have been using to keep track of information for centuries. As use of the relational model grew, its disadvantages became apparent. One of the these — relatively inefficient processing — has been largely overcome by the continuous advance of computer technology. However, processing power by itself can not overcome the inherent representational limitations of the relational model. The simplicity which is one of its most appealing characteristics essentially makes it impossible to store anything about the meaning of the data within the database itself.

In an effort to overcome this limitation, many semantic data models have been developed that capture some of the meaning, as well as the structure, of data. These models allow explicit representation of special semantic relationships, or abstractions, such as classification (instance/occurrence-of), generalization/specialization (is-a), aggregation (part-of), and association (member-of) [Brodie, 1986; Rundensteiner et al., 1994]. However, semantic data modeling concepts have not been widely incorporated into commercial database management systems or the design methodologies that support them. One reason may be the different, and sometimes confusing or contradictory, interpretations given to them. Another is the difficulty of retaining their semantic content in a relational implementation.

The objectives of this research are to:
• present a formal Data Abstraction Model that is capable of providing a unified and consistent interpretation of the widely discussed abstractions;

• propose mechanisms for representing each of the abstractions in a conventional relational database; and

• analyze the impact of these data abstractions on database design, concentrating on their implications for key selection and semantic integrity constraints.

This paper is divided into four sections. The remainder of this section provides an overview of data abstractions concepts. Section 2 presents the Data Abstraction Model that is used to define three common abstractions in terms of a uniform set of characteristics. Section 3 discusses the representation of the abstractions in the relational model. This section also explores the implications of each abstraction for semantic integrity constraints and for relational key selection. Concluding remarks are found in Section 4.

1.1 Data Abstraction Concepts

An abstraction is a simplified description or specification of a system that emphasizes some properties while suppressing others [Theodoulidis et al., 1992]. Three categories of abstraction mechanisms commonly discussed in the semantic data modelling literature are: 1) inclusion which represents the subtype-supertype relationship [Hull and King, 1987]; 2) aggregation which allows a relationship among two or more objects to be thought of as a higher-level object [Smith and Smith, 1977]; and 3) association where a collection of members is considered as a higher-level set [Brodie, 1981].

In principle, at least, each of these abstractions has specific, well-defined, semantics which can help to clarify the meaning of a database. Many researchers have pointed out the benefits, such as enhanced integrity and easier query formulation, that can be achieved by incorporating data abstractions into database management systems (for example, Costal et al. [1997]; Hruska and Kolencik [1997]; Motschnig-Pitrik and Mylopoulos [1991]; Motschnig and Storey [1995]; Papazoglou, 1995]; Goldstein and Storey [1992, 1994]; Storey [1993]). The slow integration of data abstractions into database management systems and design methodologies may be due, in part, to the fact that researchers have often used the same term to mean different things and different terms to mean the same thing. The association abstraction, for example, is sometimes referred to as membership, grouping, partitioning, or cover aggregation [Potter and Kerschberg, 1988]. Potter and Trueblood [1988] consider generalization to be simply the abstraction that involves subtypes and types whereas
Teorey et al. [1986] define *generalization* to partition the subtypes and a *subset hierarchy* to allow for overlapping subsets.

1.2 Terminology

Some of the confusion surrounding data abstractions is due to the need for terminology that simultaneously, and without confusion, describes both the real world and the database representation of that world.

**Describing the Real-World.** The real-world is composed of *things* having *properties* that are *inherent* and exist objectively whether or not they are observed or recorded [Bunge, 1977, p. 58]. A property can be characterized as descriptive or associative according to its semantic role: a descriptive property expresses some characteristic of a thing; an associative property relates two or more things. Properties can also be classified as a priori, inherited, emergent or derived:

1. An *a priori* property is a property that a thing has independently of any association(s) with other things. For example, the “date” of an *Order* is an a priori property.

2. An *inherited* property is a property of one thing that is also applicable to another thing by virtue of some association between them. For example, *Truck is-a Vehicle* implies that all properties of *Vehicle* are also properties of *Truck* [Brachman, 1983; Tanimoto, 1987].

3. An *emergent* property is a property of a composite thing\(^1\) that is not directly inheritable from any of its components, but rather *depends on* properties of the components in some, possibly complex, way. The “top speed” of a *Vehicle* is an example of an emergent property. It is not a property of any component of *Vehicle*, but is clearly determined by a complicated interaction among component properties.

4. A *derived* property is an emergent property that possesses a *known relationship* to properties of the components. The “total value” of an *Order*, for example, is a derived property that can be computed by summing the “value” properties of its component *Line Items*.

\(^1\)A formal definition of “composite thing” is given later in the section on aggregation.
Describing the Database. An entity is the database representation of a "thing" in the real world. It is useful to distinguish between an entity occurrence which represents an individual real-world thing and an entity type which represents a class of real-world things. Properties of things are represented by attributes and relationships. An attribute is the database representation of a descriptive property, whereas a relationship represents an associative property. Attributes and relationships are normally defined for an entity type. Each entity occurrence corresponding to that type will have a value for each defined attribute. For example, the attribute 'birthdate' may be defined for Student, and each occurrence of Student will have a specific value for this attribute.

The requirements of the application will determine which properties of a thing will be represented in the database. A real-world human being, for example, might be represented by height, weight, colour of eyes, and birthdate in one database, and by education, job title, and project assignment in another.

Cardinalities represent the minimum and maximum number of occurrences of a relationship in which each occurrence of an entity type can participate [Tsichritzis and Lochovsky, 1982].

\[ \begin{array}{cc}
\text{Manager} & \text{Manages} & \text{Employees} \\
1,* & & (1,1)
\end{array} \]

This states that each occurrence of Employee participates in exactly one occurrence of the "manages" relationship, whereas an occurrence of Manager participates in at least one, and possibly many (*), occurrences of "manages". Informally, every employee has exactly one manager, and a manager manages at least one employee. A discussion of cardinalities for higher-degree relationships is found in Dey, Storey, and Barron [1998].

2 DATA ABSTRACTION MODEL

This section presents a seven-dimensional model synthesized from the literature for describing and analyzing data abstractions. The model was developed from an extensive analysis of the literature and from applying abstractions to a large variety of design problems. The model, presented in Table 1, is applied here to the inclusion, aggregation, and association abstractions to provide a uniform, consistent analyses of these three widely-known abstractions. This model is also intended to be useful for evaluating new abstractions that might be proposed. The terms "sub" and "super" are used to refer to the two participants in an abstraction.
Table 1: DATA ABSTRACTION MODEL

<table>
<thead>
<tr>
<th>DIMENSION</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semantic Interpretation</td>
<td>A precise specification of the meaning of the abstraction.</td>
</tr>
<tr>
<td>Property Sets</td>
<td>Constraints on the property sets of the sub and the super, and their inter-relationship; specification of the allowed inheritances and whether the entities can have a priori, derived, and/or emergent properties.</td>
</tr>
<tr>
<td>Roles</td>
<td>When multiple subs are related to a single super, do they play distinct roles in the abstraction, or are they interchangeable?</td>
</tr>
<tr>
<td>Transitivity</td>
<td>An abstraction is transitive if the sub (super) in one instance of the abstraction can be the super (sub) in another instance of the same abstraction. If $B$ abstraction-of $A$ and $C$ abstraction-of $B$, it may be true that $C$ abstraction-of $A$, or this may be meaningless.</td>
</tr>
<tr>
<td>Mappings</td>
<td>Constraints on the allowed cardinalities, specifically, whether the minimum cardinality must or can be zero and whether the maximum cardinality must or can be greater than one. This determines whether either the sub or super can exist without the other and whether an occurrence of one of them can be related to more than one occurrence of the other. A set of abstractions, all of which involve the same super entity type, are said to form a complete covering if every occurrence of the super entity type has a corresponding occurrence of at least one of the sub entity types. If an occurrence of the super entity type can have corresponding occurrences of more than one of the sub entity types, the abstractions are said to overlap.</td>
</tr>
<tr>
<td>Existence-dependence</td>
<td>Can the sub exist independently of the super and vice versa?</td>
</tr>
<tr>
<td>Homogeneity</td>
<td>Are the sub occurrences all of the same type? Is the sub the same type as the super?</td>
</tr>
</tbody>
</table>
2.1 Inclusion

Semantic Interpretation. An inclusion abstraction, often denoted by is-a, represents a subtype / supertype relationship [Hull and King, 1987].

In a relationship, A is-a B, A is referred to as the specific entity type and B, the generic entity type. Several kinds of inclusion abstractions may be distinguished:

1. Classification is an inclusion abstraction between an entity occurrence and its corresponding entity type [Peckham and Maryanski, 1988]; for example, "Randall is-a Secretary". The inverse relationship from an entity type to its occurrence is called instantiation [Motschnig-Pitrik and Mylopoulos, 1991].

2. When a generic entity type is defined as the union of non-overlapping specific entity types, it is called a generalization; in this case, the specific entity types form a partitioning of the generic [Smith and Smith, 1977; Teorey et al., 1986]. For example, Student is a generalization of Undergraduate Student and Graduate Student.

3. Specialization is the inverse of generalization where the occurrences of a generic entity type having some distinctive characteristic are viewed as a separate subclass [Tsichritzis and Lochovsky, 1982]. In the example given above, Graduate Student is a specialization of Student.

4. Whenever more than one specific entity type is related to a single generic, it is informative to examine the relationships amongst the specifics. For example, a number of specific entity types may be defined for the generic entity type, Employee, including Secretary, Engineer, Manager, Part-Time, and Full-Time. One group of these, Part-Time and Full-Time, do not overlap and between them include all employees. A second group, Technician, Secretary, and Engineer, do not (presumably) overlap but may not include all employees. Clearly, there will be overlap between these two groups. When overlapping amongst the specific entity types can occur, a subset hierarchy is formed [Teorey et al., 1986].

Property Sets. Probably the most important characteristic of the inclusion abstraction is its property inheritance principle [Brachman, 1983] which states that anything that is true of a generic entity type such as Employee (attributes, participation in relationships, and integrity constraints) must also be true of its specific entity type, for example, Secretary [Korth and Silberschatz, 1991].

The major significance of property inheritance in the database context is that the common properties need only be represented once at the highest, most generic, level and can be applied automatically to all lower levels, thereby minimizing storage requirements and reducing redundancy.

Emergent properties are relevant only to situations where one entity type is comprised of other, component, entity types. Since this is not the case for inclusion, there can be no emergent properties. Furthermore, by definition of the inclusion abstraction, the specifics inherit all of the generic's
properties, so the existence of any property of the generic that was not also a property of the specific would be a contradiction.

Roles. In the inclusion abstraction, the individual specific entity types do not perform particular functions. Thus, they do not have distinct roles.

Transitivity. The inclusion abstraction is transitive [Tanimoto, 1987]. If, for example, Secretary is-an Employee and Employee is-a Person, then, Secretary is-a Person.

Mappings. The cardinalities for an inclusion relationship are always: Specific (1,1) is-a Generic (0,1). That is, for each occurrence of the specific there is exactly one occurrence of the generic. An occurrence of the generic may or may not have a corresponding occurrence of the specific.

Homogeneity. The specifics related to a single generic need not be homogeneous. For example, Engineer and Secretary can both be related to Employee. The two entity types in an inclusion are homogeneous to the extent that all of the specifics related to a given generic must have all of the generic’s properties.

Existence-dependence. In any inclusion abstraction, the specific entity type is dependent upon the existence of its generic. That is, an occurrence of the specific can not exist unless the corresponding occurrence of the generic exists. This is obvious from the fact that the cardinalities of the specific are (1,1). The generic, in general, is not dependent upon the specific for its existence. However, in the case of a set of inclusions forming a complete covering, such as a generalization, the existence dependency occurs in both directions.

2.2 Aggregation

Semantic Interpretation. Aggregation is an abstraction in which a relationship between objects is considered as a higher level (aggregate) object [Smith and Smith, 1977]. When considering the aggregate, specific details of the constituent objects are suppressed. Formally, an aggregate is a subset of the cartesian product of its components [Codd, 1979]. Reservation, for example, is an aggregation of Hotel, Person, and Date [Smith and Smith, 1977]. It can have its own attribute, as
Figure 1: Aggregation Abstraction

well as participate in other relationships. When necessary, the details of a particular reservation can be retrieved via the appropriate component-of relationship; for example, Person component-of Reservation.

The aggregation example given in Figure 1 is based on Brodie [1981]. This notation (a horizontal line over the components with a directed line to the aggregate) is adopted throughout the remainder of this paper to represent aggregation abstractions. Committee is an aggregation of the attribute “name”, the entity types Chairperson and Secretary, and the set entity type Membership. The notion of a set entity type is explained in the next section on association.

There are three kinds of aggregation:

1. An attribute can be an aggregation of other attributes [Hull and King, 1987]. For example, the attribute “address” is the aggregate of “number”, “street”, “city”, “PO box#”, and “zip code”.

2. An entity can be an aggregation of entities and/or attributes, such as the Committee example of Figure 1.

3. A relationship can be an aggregation of entities and attributes, such as the previously mentioned Reservation. Similarly, Assignment could be an aggregation of the entity types Employee and Project and the attributes “start-date” and “end-date” as shown in Figure 2. Of course, a relationship can be an aggregate of more than two entity types; for example, Teaching-Assignment could be the aggregate of Instructor, Session, and Room, along with “start-time” and “end-time”.

The components of an aggregate can be classified into a three-level hierarchy [Dos Santos et al., 1980; Biller and Neuhold, 1978]. A component that is permissable but not required is referred to as relevant. A relevant component is characteristic if its presence is required for the existence of the
aggregate, and a characteristic component is identifying if it uniquely identifies the aggregate. For example, Air-conditioning is a relevant component of Car, Steering-wheel is a characteristic component, and Engine is an identifying component. In order to ensure the integrity of an aggregation, it should not contain any components that are not relevant, it must contain each characteristic component, and it should not contain more than one instance of any identifying component.

Property Sets. Aggregation is usually described as having partial inheritance from the components to the aggregate [Potter and Kershberg, 1988]. For example, if Car is an aggregation of Engine, Steering-Wheel, etc., and “horsepower” is an attribute of Engine, then “horsepower” is also an attribute of Car. Partial inheritance from the aggregate to its components is also possible for entities or relationships that are aggregates of entities and attributes. In the case of a physical aggregate, such as Car, the components may be said to inherit the value of their “location” attribute from the aggregate. Similarly, the Line.item component of an Order can inherit “delivery date”. Partial inheritance does not apply to attributes that are aggregates of attributes.

Emergent properties, by definition, can apply only to the aggregate. For example, “top-speed” is an emergent property of Car. It is not a property of any component of Car, although it is obviously dependent on the properties of the components of Car.

Roles. Each of the component entity types plays a specific role in the aggregate. For example, Seat as a component of Car plays a particular role that cannot be played by Wheel, Engine, or any other component. In other words, the parts are not, in general, interchangeable. The
situation where an aggregation contains a number of equivalent, interchangeable components, e.g. the members of a committee, is dealt with by defining a set entity type.

**Transitivity.** Transitivity holds for the aggregation abstraction; that is, if A part-of B and B part-of C, then one can conclude that A part-of C. For example, if Spoke part-of Wheel and Wheel part-of Bicycle, then Spoke part-of Bicycle. Note, however, that some qualification may be required. Given Switch part-of Radio and Radio part-of Car, then Switch part-of Car is certainly true, but to properly understand this relationship, it is necessary to know the complete chain of aggregations.

**Mappings.** The allowed mappings for each of the three kinds of components are most easily described by use of min/max cardinalities:

1. **Relevant:**

   Relevant Component-of Aggregate
   \[
   (-, -) \rightarrow (0, >0)
   \]

   A relevant component is optional so the minimum cardinality of the aggregate is ‘0’. The maximum cardinality must be greater than zero in order to avoid aggregations that are semantically absurd such as:

   Air Conditioning component-of Bicycle
   \[
   (-, -) \rightarrow (0, 0)
   \]

2. **Characteristic:**

   Characteristic component-of Aggregate
   \[
   (-, -) \rightarrow (\geq 1, \geq \min)
   \]

   For a characteristic component, the minimum cardinality of the aggregate is ‘\( \geq 1 \)’ to reflect the fact that such components are required. The maximum cardinality, as always, must be greater than or equal to the minimum cardinality.

3. **Identifying:**

   Identifying component-of Aggregate
   \[
   (-, 1) \rightarrow (1, \geq 1)
   \]
An identifying component implies, first, that the component must exist so the minimum cardinality of the aggregate is ‘1’. In order for the component to identify uniquely the aggregate, the maximum cardinality of the component must be ‘1’.

In each of the above cases, the minimum cardinality of the component entity type may be ‘0’ or ‘1’. If the component entity type can exist independently of its attachment to its aggregate, then its minimum cardinality is ‘0’ [Blaha et al., 1988]. If, however, the component is considered only in connection with its aggregate, then, its minimum cardinality is restricted to be ‘1’. To an auto parts supplier, engines, wheels, etc. can exist without being attached to a car. To a parking garage, however, they exist only as parts of cars.

**Homogeneity.** The components of an aggregate can be heterogeneous, and the components are usually of a different type than the aggregate. There can, however, be recursive aggregations such as *Part component-of Part*.

**Existence-dependence.** Components can always exist without requiring the existence of the aggregate. If the aggregate exists, however, any characteristic or identifying components must also exist.

### 2.3 Association

**Semantic Interpretation.** Brodie [1981] describes *association* as a form of abstraction in which a collection of members is considered as a higher-level (more abstract) set, called an entity set type. The details of the member objects are suppressed and properties of the set object (the association) emphasized. In the *Committee* example of the previous section, *Membership* is an association of the *Person* entity type. This is denoted as shown in Figure 3. The association can also participate in relationships (for example, *Membership component-of Committee*).

*Aggregation* and *association* are both mechanisms for treating a collection of objects as a higher-level object. There are, however, notable differences between them:

1. The members of an association are all of the same kind, whereas the components of an aggregate can be, and usually are, of different kinds [Brodie, 1981, p.603].

2. The association abstraction allows for an indefinite number of members. *Aggregation*, in contrast, requires that the number of components be fixed.
Figure 3: Association Abstraction: Person to Membership

3. Each component in an aggregation plays a specific role whereas the members of an association are interchangeable with respect to the set.

In the following example of association, it is assumed that there are various Frequent Flyer Plans to which a person could belong, and that all share a common set of attributes.

*Member Entity Type:*
*Person:* [SSN, ...]

*Set Entity Type:*
*Frequent-Flyer-Plan:* [AIRLINE, #.of.participants, ...]

*Association Abstraction:*
*Person member-of Frequent-Flyer-Plan:* [#bonus.miles, #free.trips.taken]  
\[(0,*)\]

The set entity type, Frequent-Flyer-Plan, has an a priori attribute, “airline”, and derived attribute, “# .of.participants”. Since there can be multiple Frequent Flyer Plans, the key of the set entity is “AIRLINE”.

**Property Sets.** Unlike the previous abstractions, there is no inheritance of properties in the association abstraction [Potter and Kerschberg, 1988]. The association entity type can, however, have derived properties; for example, the “average salary” of a set of employees. It can also have a priori properties such as the maximum number of members of a committee.

**Roles.** The member entity occurrences are not required to play distinct roles in an association. With respect to the set occurrence, they are interchangeable.

**Transitivity.** The association abstraction is transitive. For example, Paper member-of Session and Session member-of Conference-Program implies that Paper member-of Conference-Program.
Mappings. The mapping between member and set entity types is unrestricted; a member can belong to any number of sets and a set can have any number of members. This is expressed in the following cardinalities:

\[
\text{A member-of B} \\
(0, *) \\
(0, *)
\]

If the application semantics dictate that it is not meaningful for the set to exist without having at least one member, then the cardinalities of the set entity type are \((1, *)\).

Homogeneity. The member entity occurrences are homogeneous amongst themselves. The set entity is necessarily of a different kind.

Existence-dependence. Neither the member nor the set is dependent on the other for existence.

2.4 Summary

Table 2 summarizes the results of the above analysis. Each column presents a convenient summary of one of the common data abstractions in terms of the model developed in this paper. In addition, the table allows easy comparison of the abstractions along each of the model’s dimensions. Since the three abstractions have unique semantics, one should not necessarily expect to find much similarity among them.

Property Sets. Not surprisingly, both the sub and super in all three aggregation can have a priori properties. Downward inheritance is a characteristic of inclusion; aggregation can have partial inheritance in both directions. Emergent and derived properties can apply only to aggregation and association since they require that the relationship be characterizable as one between a “whole” and its “components”.

Transitivity. All three abstractions are transitive.

Mappings. Only inclusion has definitive cardinalities. However, cardinalities can be used to distinguish among the three types of aggregation. A sub in an inclusion abstraction is related to exactly one instance of the super. In association (and in aggregation except for identifying components), a sub can be associated with many supers.
### Table 2: DATA ABSTRACTIONS: SUMMARY

<table>
<thead>
<tr>
<th>ABSTRACTION</th>
<th>INCLUSION</th>
<th>AGGREGATION</th>
<th>ASSOCIATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semantic</td>
<td>Sub is-a</td>
<td>Sub part-of</td>
<td>Sub member-of</td>
</tr>
<tr>
<td>Interpretation</td>
<td>Super</td>
<td>Super</td>
<td>Super</td>
</tr>
<tr>
<td>Property Sets</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• A Priori</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>• Inherited</td>
<td>✓</td>
<td></td>
<td>Partial</td>
</tr>
<tr>
<td>• Emergent</td>
<td>—</td>
<td>—</td>
<td>✓</td>
</tr>
<tr>
<td>• Derived</td>
<td>—</td>
<td>—</td>
<td>✓</td>
</tr>
<tr>
<td>Do Subs have distinct roles?</td>
<td>—</td>
<td>✓</td>
<td>—</td>
</tr>
<tr>
<td>Transitivity</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Mappings</td>
<td>(1,1)</td>
<td>(0,1)</td>
<td>(0,*)</td>
</tr>
<tr>
<td></td>
<td>• Rel. (<strong>,</strong>)</td>
<td>• Rel. (0,&gt;0)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Char. (<strong>,</strong>)</td>
<td>• Char. (≥1,≥min)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Id. (__,1)</td>
<td>• Id. (1,≥1)</td>
<td></td>
</tr>
<tr>
<td>Homogeneity:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Among Subs</td>
<td>—</td>
<td>—</td>
<td>✓</td>
</tr>
<tr>
<td>Subs - Super</td>
<td>One Way</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Existence</td>
<td>✓</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Dependencies</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Rel. —</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Char. ✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Id. ✓</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Homogeneity. Homogeneity among the sub instances is an essential characteristic of association. A form of homogeneity exists for inclusion in that each sub instance must have a corresponding super instance.

Existence Dependencies. An instance of the sub in an inclusion can only exist if its corresponding super instance exists. In aggregation, existence of a super instance depends on the existence of instances of any characteristic or identifying components.

3 USE OF ABSTRACTIONS WITH RELATIONAL DBMS

This section examines the use of data abstractions in a standard relational DBMS. It considers both representation issues and implied semantic integrity constraints.

Abstractions in the Relational Model. A data abstraction will be correctly, and completely, represented by a relational structure if all of its syntactic and semantic characteristics are captured accurately [Markowitz and Shoshani, 1989]. The relational model can always provide a precise representation of the syntactical aspects of an abstraction (i.e., it represents each entity, relationship, and attribute value). However, it is unable, on its own, to represent the semantics of an abstraction. Thus, it is necessary to supplement the relational structure with additional information, typically in the form of integrity constraints and triggered procedures. That is, constraints that are implicit in the conceptual model must be expressed explicitly in the relational one [Troyer, 1989].

The columns of a relation may be labeled to suggest their meaning. However, the relational data model contains no formal mechanism for recording or processing semantic information. For example, if two relations each contain a column labeled “employee-number”, one might infer that they represent the same thing, but there is no means of knowing for certain. Similarly, it may be that a column “employee-number” in one relation represents the same real-world property as the column “ID” in another relation, but in this case, even the labels do not match. Thus, representing any data abstraction in a relational database requires two steps. First, a relational representation must be found that is compatible with the intended semantics. Then, the knowledge about exactly what this representation means must be explicitly represented — in some combination of a data dictionary entries and procedures that are invoked when the database is processed. The remainder
of this section outlines how this might be accomplished.

3.1 Implementing Abstractions

The current version of the Oracle DBMS is used as the foundation on which to construct implementations of the various data abstractions. The selection of Oracle was based on its leading position in the relational DBMS marketplace. Because a complete description of the implementation is several times as long as this entire paper, it is only possible to outline the general approach and give a few examples.

There are two keys to implementing data abstractions in Oracle: the ability to define views on the actual database and the use of the USER_TAB_COMMENTS and the USER_COL_COMMENTS tables in the data dictionary to document the results. Built-in integrity features such as REFERENCES and ON DELETE CASCADE are valuable in many situations. For some more complex operations, it is necessary to use triggers and macros in PL/SQL. Implementing the abstractions was tedious but not particularly difficult, and the complexity is not visible to the database users. The most serious potential problem arises from the extensive use of the view mechanism. This has a small negative impact on processing efficiency, but more importantly, views defined on more than one base table can not be updated. Thus, while the mechanisms described are fully functional for insertion, deletion and retrieval, they may not all work for update.

3.2 Inclusion

3.2.1 Compatible Relational Representations

Classification. The representation of a classification relationship, for example, “Diane Stone is-a Manager” is implicit in the instantiation of an entity type as a set of occurrences.

Specialization. If the specific entity type has no unique properties that are to be included in the database, the specialization relationship can be represented by a logical-valued attribute of the generic entity type indicating whether each of its occurrences belongs to the specific entity type.

E.g.:
Specialization: Manager is-an Employee
Given the entity:
Employee: [EMP#, dept, title, ...]
Specialization represented by a logical-valued attribute of generic:
Employee: [EMP#, dept, title, ..., manager?]  

If, on the other hand, it is necessary to represent properties of the specific entity type other than those that are inherited from the generic entity type, a separate relation must be created for the specific and linked to the generic entity type. The most direct way to make this link is to add the key of the generic as a foreign key attribute of the specific. In the common case where the specific and generic entity types have the same key, an additional column is not needed, but an entry must be placed in the USER.COL.COMMENTS table to record the fact that the primary key is also a foreign key.

Specialization represented by a foreign key (which is identical to a local key):
Manager: [EMP#, budget, project, ...]

This solution is also available for the case where there are no unique properties of the specific entity type to be represented. The two representations will have significantly different performance characteristics. The choice should be based on the proportion of generic entity type occurrences that belong to the specific entity type and on the types of queries that are to be processed.

The need for explicit representation of the specific entity type is obvious when it has unique attributes. This solution is also required if there are relationships that involve only the specific entity type. Suppose, for example, that only managerial employees participate in a stock option plan. It is not sufficient to just include an attribute representing stock option plan participation in the Employee entity type because: 1) space would have to be allocated for this attribute, even for employees to whom it does not apply; and 2) explicit attention would have to be given to enforcing the integrity constraint that only full-time employees could have a stock option plan. The preferred solution is to create a separate entity type, Manager, having a foreign key attribute Stock.Option.Plan_ID.

Generalization. There are two alternatives for representing a generalization abstraction, depending upon whether the specific entity types have unique attributes. If they do (which is the more likely case), then they must, of course, be represented by distinct relations. If no unique attributes of the specifics need to be stored, a discriminating attribute in the generic entity type can be used to distinguish among the specifics. This discriminating attribute is similar to the
logical-valued attribute used to represent specialization except that its value identifies the specific
to which each instance of the generic belongs.

E.g.:
Employee is the generalization of Engineer, Secretary, and Technician
can be represented by:
Employee: [EMP#, name, ..., employee-type]

where "employee-type" must take one of the values 'engineer', 'secretary', or 'technician' for a
particular employee occurrence. This representation is possible because the specific entity types
are non-overlapping. Of course, if any of the specific entity types have attributes of their own, a
separate relation is required in addition to the discriminating attribute. If all of the specifics have
unique attributes, the discriminating attribute can be omitted, although retaining it may enable
processing efficiencies.

Subset Hierarchy. In a subset hierarchy, there may be overlap amongst the specific entity
types; for example [Teorey et al., 1986], Full-Time is-an Employee, Engineer is-an Employee, and
Secretary is-an Employee. In this case, there must be two discriminating attributes in the generic
entity type — a logical-valued one for "full-time" and another for "type-of-employee". In general,
the number of discriminating attributes will equal the number of potentially overlapping specific
entity types. Use of a single discriminating attribute to distinguish all of the possible combinations
of the overlapping subtypes (that is, one code for each of Manager, Engineer, Manager-Engineer,
etc.) is feasible in principle, but would lead to excessively complex search criteria.

3.2.2 Integrity Constraints

Integrity constraints implied by the inclusion abstraction are:

- There must be an occurrence of the generic entity type for each occurrence of any of the
  specific entity types.

- In a generalization, every occurrence of the generic entity type must have a corresponding
  occurrence of exactly one of the specific entity types [Teorey et al., 1986].

- Each discriminating attribute is restricted to a set of allowed values.

- The discriminating attribute in a generalization abstraction cannot be null.

- If a generic entity occurrence is deleted, its corresponding specific entity occurrence(s) must
  also be deleted.
• If a generic entity occurrence is added in a generalization, its corresponding specific entity occurrence must also be added.

• If a specific entity occurrence is deleted (added) in a generalization hierarchy, the generic entity type must also be deleted (added) [Hull and King, 1987].

3.2.3 Relational Implementation.

In general, any subset-superset relationship can be represented by including the key of the super as an attribute of the sub and making it behave as a foreign key by specifying that it be NOT NULL, that it REFERENCES the primary key of the super, and (via the ON DELETE CASCADE option) that the sub occurrence must be deleted when that of the super is.

For a generalization, ensuring that the discriminating attribute has an appropriate value can be done with a constraint of the form CHECK (discriminating_attribute IN (list)) along with a NOT NULL constraint.

Perhaps the most important property of the inclusion abstraction is attribute inheritance. This is implemented using views. For example, given the base relations:

Employee (EMPNO, name, salary)
Secretary (EMPNO, data-entry_speed)

it is possible to define Secretary_view by:

CREATE VIEW secretary_view
AS SELECT empno, name, salary, data-entry_speed
FROM employee, secretary
WHERE employee.empno = secretary.empno

This type of construction can be cascaded as needed to achieve multiple levels of inheritance. With some reasonably simple manipulation, it is possible to retrieve the structure of inheritance relationships from the data dictionary table USER_CROSS_REFS and to record what has been done using the COMMENT ON command to insert a tuple in the USER_TAB_COMMENTS table. Because a sub occurrence can not be created unless its corresponding super occurrence already exists, a simple PL/SQL procedure "insert_secretary" can be defined that accepts data values from the user and creates both the Employee and Secretary tuples.
3.3 Aggregation

Some brief comments on the relational implementation of aggregation are included in the following sections on representation and integrity constraints.

3.3.1 Compatible Relational Representations

There are three types of aggregation to be considered:

1. an attribute is an aggregation of other attributes

   Example: It may sometimes be advantageous to use the aggregate attribute “address” instead of dealing explicitly with all of its components: {PO-box, number, street, city, state, country, postal code}. This is easily implemented by creating a view with an address attribute that is defined as the concatenation of the individual address components in the base relation. This would be documented in the USER_COL_COMMENTS table. It is actually possible, via a PL/SQL procedure, to retrieve or update components of an address referenced in this way as long as either the components are of known length or separated by a known delimiter.

2. an entity is an aggregation of attributes and other entities

   The aggregate is defined simply as a view on the tables (or views) for each of the entity components. The situation is complicated somewhat by the need to allow for optional (or multiple) components and components that are attributes rather than entities. However, all of these cases can be handled with appropriate macros.

3. a relationship is an aggregation of entities and, possibly, attributes

   A relationship, considered as an aggregation, is represented in the same way as an aggregate entity.

3.3.2 Integrity Constraints

Integrity constraints implied by an aggregation abstraction are:

- An aggregate is, inherently, something that is made up of components. If any of its characteristic components do not exist, the aggregate cannot exist. This is implemented using REFERENCES and ON DELETE CASCADE.
• The number of components is implicit in the definition of the aggregate, and each component plays a specific role (for example, "left front wheel", "chairperson", etc.).

• Any components that do not exist independently of the aggregate, should be deleted whenever the aggregate is deleted. For example, there should not be any occurrences of the Chairperson entity type that are not related to a Committee. This is the reverse of the first constraint above. However, the implementation mechanism is the same.

3.3.3 Entity Identification.

The aggregation abstraction has implications for the identification of both an aggregate and its components. If an entity type is an aggregation of attributes and/or other entity types, then instances may be identified by the value(s) of identifying attributes, and/or the key value(s) of identifying entity types.

In some situations, component entity types can be identified independently of their aggregate. An engine of a car, for example, can have a unique "Serial Number". In other situations, the identity of the aggregate is needed to distinguish amongst occurrences of the component and it might even be sufficient.

• If there is only one of a certain component related to the aggregate, then the key of the aggregate is sufficient to uniquely identify the component and can, thus, serve as the component’s key. For example, identifying a car implicitly identifies its steering wheel.

• When there are multiple occurrences of a component for a single aggregate, each one may be identified by the key of the aggregate plus something that distinguishes among them, e.g. role or location ("left-front" tire of a car).

When a relationship is thought of as an aggregation, how it is identified depends upon the cardinalities of the component entity types.

• At Least One Entity Type has (1,1) Cardinalities

A component entity type having (1,1) cardinalities uniquely identifies the relationship aggregate. For the relationship Teaching-Assignment defined as the aggregate {Instructor, Section}, we have:
Instructor component-of Teaching-Assignment
(0,*)                  (1,1) (Characteristic)
Section component-of Teaching-Assignment
(1,1)                  (1,1) (Identifying)

The key of Section can serve as the key of Teaching-Assignment:

Teaching-Assignment: [SECTION_KEY, instructor_key]

If two (or more) of the component entity types have (1,1) cardinalities, the key of either can serve as the key of the aggregate. The choice is determined by the types and frequencies of anticipated queries.

- No Entity Type has (1,1) Cardinalities

In this case, no single component entity type uniquely identifies the relationship aggregate; however, some combination of two or more of them must be identifying. For the relationship Enrollment defined as the aggregate {Student, Course}:

Course component-of Enrollment
(0,*)                  (1,1) (Characteristic)
Student component-of Enrollment
(0,*)                  (1,1) (Characteristic)

The concatenation of the keys of Course and Student is needed for the key of Enrollment:

Enrollment: [COURSE_KEY, STUDENT_KEY, grade]

If there are more than two component entity types, some identifying combination of them must be used.2

3.4 Association

3.4.1 Compatible Relational Representations

An association abstraction is actually represented by three relations: one containing the occurrences of the set entity type and their a priori and derived properties; one containing the occurrences of the member entity type; and one (representing the actual association) relating specific member

2Of course, the enrollment concept could be expanded to capture the fact that a student could enroll in a course more than once by including information on the semester/quarter in which the student enrolled.
occurrences to the appropriate sets. Since, in general, a "member" can belong to more than one "set" and a set can have more than one member, the relational representation of association is:

\[
\text{Association: } [\text{set-id, member-id, [attributes]}]
\]

The ON DELETE CASCADE option is used to ensure that corresponding occurrences of the third (association) relation are deleted whenever instances of the set or member relations are deleted.

**Semantic Integrity Constraints** Any attributes of a set entity type that are derived from attributes of its members must be adjusted whenever the membership changes or whenever the relevant attribute values of the members change. This is accomplished by a triggered procedure that updates the set attribute whenever a member occurrence is inserted, deleted, or changed.

4 CONCLUSION

Although data abstractions are often cited in the literature as important modelling concepts, their presentations are often incomplete or inconsistent. This paper has defined and analyzed inclusion, aggregation, and association, using a seven-dimensional general model of an abstraction. The use of the model highlights similarities and differences among the abstractions and ensures that they are treated in a uniform, consistent way.

Despite their evident value, data abstractions are not widely used in practice, largely because they are not supported by popular commercial database management software. This paper has outlined a relational representation for the common abstractions which does not require any capabilities beyond those found in the current release of the Oracle DBMS. Although some additional work is required during database design to explicitly represent the meaning and implications of the abstractions, the result is transparent to the actual users.
5 REFERENCES


