Information Systems Vol. 14, No. 2, pp. 141–149, 1989
Printed in Great Britain. All rights reserved

0306-4379/89 $3.00 + 0.00
Copyright © 1989 Pergamon Press plc

FITTING REALITY INTO A DATABASE MOLD: RULES FOR DATA COLLECTION

RAYA FIDEL and MICHAEL CRANDALL
Graduate School of Library and Information Science, University of Washington, Seattle, WA 98195, U.S.A.

(*Received* 27 *January* 1987; *in revised form* 7 *November* 1988)

Abstract—Presents a typology of rules for database design through examination of one such set of rules, the *Anglo–American Cataloguing Rules*, Second Edition (AACR2). Analysis of the AACR2 reveals three clusters of rules: (1) content rules; (2) format rules; and (3) data collection rules. The last cluster is of interest to database designers in general, and includes the following types of rules: (1) rules for authorized sources of information; (2) rules for establishing entities, relationships and attributes; (3) rules for domains; (4) rules for occurrence; and (5) rules for cardinality. Further examination of these rule types shows particular instances which require specialized kinds of rules to resolve cases of absent, multiple, fuzzy or ambiguous data. Though useful for database design work as it stands, the typology could be enriched by addition of rule types from other databases and by further exploration of relationships between rule types and data characteristics.

"An information system (e.g. data base) is a model of a small, finite subset of the real world." [1]

1. INTRODUCTION

Developing a database involves four distinct functions, each building on the previous one: analysis, design, coding and testing. In the *analysis* stage, the database designer defines what a database should do to make it most useful to potential users by studying user needs (data requirements analysis) and documentation. During *design*, the process that defines how a database will perform its tasks, the designer concentrates on software and hardware considerations, *coding* represents the actual implementation of the design. *Testing* is carried out before the full-scale installation of a database to examine how well it will perform.

This article relates to the first function—analysis. In this stage, database designers are building a formal model of that part of the real world which will be represented in the database (the *enterprise*). This model includes a formal presentation of the components that should be included in the database and their relationships, as well as a data dictionary that includes information about each individual component, such as its definition and relations to other components. Both the formal model and the data dictionary are part of the conceptual schema of a database.

For a database to be useful to its potential users, this model must be derived from data requirements of potential users. For example, consider a database that provides information about restaurants. Suppose also that the database is designed to support a variety of functions such as inspecting and licensing restaurants, as well as going to restaurants or reviewing them. Each such function defines a purpose for which the database is going to be used, and a group of potential users who are interested in that purpose. Each such purpose and the relevant user group create an *environment* for the database.

The restaurant database in our example has two environments. The first is composed of the city personnel who will consult the database for the purpose of inspecting and licensing restaurants—the administration environment. The second is created by all users who will turn to the database when they plan to use the services offered by restaurants—the clientele environment. To build a database, then a designer must analyze the data requirements of users in each environment.

The administration and clientele environments are different in nature. The first represents a *structured environment* because the process of inspecting and licensing a restaurant is already established and has a well-defined structure. The clientele environment, on the other hand, is an *unstructured environment* because the process of deciding which restaurant to go to, or which one to review, has no apparent inherent structure.

Data requirements analysis of structured environments is conducted using methods of structured system analysis; these are well-documented in the database literature [2]. The analysis of data requirements of unstructured environments is less developed; Cerri and his colleagues [3] proposed a method that is applicable to such environments—a method that was further developed by Fidel [4]. This method uses the entity-relationship diagram [5] to represent the components to be included in a database and their relationships.

The data dictionary is constantly updated as the formal model is created. When the model is completed, designers should add instructions to the data dictionary for the personnel who will collect the actual data, explaining how to interpret real objects and facts in terms of the database. These types of guidelines are not addressed explicitly by the database literature. In this paper we propose a typology of rules that can guide database designers in developing guidelines for data collection. This typology was developed by the authors as part of a systematic approach they proposed for the construction of the conceptual schema [4].

2. THE PROBLEM

As we have pointed out above, to guarantee a reliable and consistent representation of reality in a database, data collectors need guidelines or rules, that tell them how to control the diversity they face in the real world. Moreover, even when the components of a database are clearly and rigorously defined, data collection encounters many instances when an individual object or a fact is an exception to the rule. Consider a restaurant database. A problem in selecting a name for a restaurant arises if the sign in the window says "Julia's" but the telephone directory calls it "Julia's on the Ave."

If the restaurant database is being developed only for a city's licensing department, it is likely to include a rule that would tell data collectors how to determine a restaurant's name. City personnel had probably encountered similar situations in the past and have provided pertinent instructions. When data requirements are analyzed, these instructions would be "discovered" by the designers and would be added to the definition of the component *name of restaurant*.

In other words, if a database is designed to replace an existing information system, rules to control diversity that is presented by reality already exist and they are incorporated into the database schema. When a database is created as a new information system, however, such rules are not available to designers. If the restaurant database is designed to answer questions about dining places for instance, it is the responsibility of the designers to anticipate irregularities or exceptions and to devise rules to manage them.

The literature about database design rarely discusses these rules explicitly. This is no surprise because most tools for data requirements analysis are designed to analyze existing systems, rather than to create a database for a disorganized or chaotic reality. One is left to speculate that the literature assumes that the rules would emerge from data requirements analysis.

Moreover, one may claim that it is pointless to deal with a set of rules to manage diversity on a general level because each database would require its own rules, depending on the reality it is designed to represent. This assertion, however, is not the whole truth. Although each database requires its own set of rules, *types* of rules are common to many databases. The assumption that the types of such rules can be discovered, is the motive force behind the project reported here.

The purpose of our project was to build a typology of rules to manage diversity during data collection. Such a typology can serve as a checklist for database designers, pointing to possible exceptional situations that may emerge in data collection.

3. THE METHOD

Our approach to uncovering the types of rules was to analyze an existing set of rules and to define the kinds of problems they anticipate. For that purpose we selected the *Anglo–American Cataloguing Rules*, Second Edition (AACR2) [6] which guide the description of library materials. These rules instruct catalogers how to represent library items on catalog cards.

We choose to examine the AACR2 for two reasons. First, although not always recognized, libraries have a long experience in database design. Library catalogs are databases that provide a central store of data—bibliographic data—for multiple uses. Principles of representing library materials in rigorously structured records were first developed a century and a half ago, and they have been expanding during the years as more experience in collecting bibliographic data was gained. Thus, the AACR2 represent a documentation of a long experience in the collection of data and the design of rules for recording that data.

Second, the AACR2 have to deal with a world of large diversity. Library materials range from books through machine-readable data files to 3-D artifacts. Publishers of these materials are free to ignore any information they wish: a book may lack a title, the name of the creator of a data file may be unknown, or a book may be published by any number of publishers. As a result, exceptions are more common than the rule. The AACR2 deal with this diversity to ensure the consistency and reliability of bibliographic records.

In sum, the AACR2 are useful for analysis because they are the result of a long experience in data collection in a world with large diversity.

The AACR2 consist of two parts. The first deals with the descriptions of bibliographic items, i.e. the creation of bibliographic records, and the second with the creation of access points, i.e. search keys. The first chapter includes general rules for the creation of records, and the remaining chapters in this part address specific types of materials, such as books, maps or data files. For our project we analyzed the general rules in the first chapter only.

With the entity-relationship model [5] as a theoretical framework, we examined each rule in the first chapter to determine the type of question it addresses. The rule that instructs, for instance, how to determine the place of publication when a number of cities

1 RULES FOR AUTHORIZED SOURCES

- **1.1 General rules**
- **1.2 Lack of authorized source**
- **1.3 Multiple authorized source**

2 ESTABLISHING ENTITIES, RELATIONSHIPS, AND ATTRIBUTES

- **2.1 Elaboration rules**
- **2.2 Fuzzy entities, relationships and attributes**
    - **2.2.1 Fuzzy entities**
    - **2.2.2 Fuzzy relationships**
        - Procedure to establish relationships
        - Upper limits on relationships
        - Relationships that cannot be established
        - Doubtful cases
        - Erroneous information
        - Multiple correspondence
    - **2.2.3 Fuzzy attributes**
        - Same as fuzzy relationships

3 RULES FOR DOMAINS

- **3.1 General rules**
    - Basic rules
    - Content rules
    - Missing-value rules
    - Sequence rules
    - Shortcut rules
    - Quality rules
    - Authority lists
- **3.2 Fuzzy domains**
    - Lack of correspondence
    - Incomplete information
    - Invalid values
    - Too many values
    - Indistinguishable values
    - Unreliable data
    - Values from other domains

4 OCCURRENCE

5 CARDINALITY

Fig. 1. The typology of rules.

appear on an item answers the question of what city qualifies as "place of publication." The *type* of question this rule defines is: how to determine whether or not a certain real-life object or fact qualifies as an entity.

Next, question types were grouped into more general types of rule. The type of rule that emerges from the example is: *rules for establishing entities*. In fact, this example represents a specific case of the rule type: rules for establishing entities when more than one object or fact in the real world qualifies as an entity.

Figure 1 summarizes the typology of rules for data collection developed from our analysis of the AACR2. In the following text we have used examples from a restaurant database rather than the AACR2, since this will be more familiar to most readers; it also shows the applicability of the rule types in another situation.

4. A TYPOLOGY OF RULES

The rule types in AACR2 create three clusters. The first, which we call "content rules," defines the elements in a bibliographic record. In the entity-relationship terminology this cluster defines the components in the entity-relationship diagram for a bibliographic description. While important for initial design work, these rules are not critical for data collectors except to define the boundaries of their data "world." In a restaurant database these rules would determine, for instance, what information about owners of restaurants, or about buses going to restaurant, to include in the database.

The second cluster, which we call "format rules," instructs catalogers how to record bibliographic data. Although important for library work, these rules are on the internal level of design and, again, are of no interest to our project. In a restaurant database such rules would be used to determine what delimiters separate data entries for buses going to the same restaurant, or whether the price should be recorded in a fixed or variable field.

The third cluster includes several types of rules that are pertinent to data collection in general (see Fig. 1). These are: (1) rules for authorized sources of information; (2) rules for establishing entities, relationships and attributes; (3) rules for domains; (4) rules for occurrence; and (5) rules for cardinality.

4.1. Rules for authorized sources

A special set of rules must be created to determine which sources of information to consult about entities (i.e. entity types), relationships or attributes. For example, what sources of information should be consulted to determine the name of a restaurant? Should the name be taken from the restaurant's sign, from the telephone directory, or from the license of the restaurant? Obviously, only one source should be used, and a rule is needed to state clearly what the authorized source of information is.

Rules for authorized sources are of three types: (a) a general rule that states the source of information; (b) a rule for individual entities, relationships or attributes that lack the authorized source; and (c) a rule for those that have more than one authorized source.

4.1.1. General Rules

A general rule clearly states the authorized source of information, and can take one of two forms. Either a database designer selects one, and only one, source or he decides that *any* source will do. Thus, a designer may instruct data collectors to record the license number from the license itself, but to use any source of information—the chef, employees or diners—to determine the type of food that is served.

Attributes may require a special form of a general rule: a rule for dependent authorized sources. Such a rule states that the authorized sources for information for an attribute is the same one that is used to establish the entity or relationship which it describes. This kind of a rule implicitly covers instances where the source is not available, or those where there are more than one source.

4.1.2. Lack of Authorized Source

If the general rule designates a *single* authorized source of information, a rule must be devised for individual entities, relationships or attributes that lack such a source. Such a rule can take one of two forms: it may allow data collection from *any* other source or it may provide a list of other sources that can be used, and these sources may be organized in order of preference.

4.1.3. Multiple Authorized Sources

In contrast, an individual entity, relationship or attribute may have more than one authorized source of information. To solve such a problem a database designer can designate additional criteria by which the authorized source of information should be selected. Suppose the menu is the source used to determine the type of food served in a restaurant. In anticipating a situation when a restaurant has more than one menu, one may instruct data collectors to consult the menu printed most recently, and if all menus were printed at the same time, to consult the one that includes the largest number of dishes.

4.2. Establishing entities, relationships and attributes

Rules to establish entities, relationships or attributes guide data collection when it is not clear whether or not a particular object or a fact in real-life "qualifies" as a specific entity, if a relationship exists among real-life objects, or whether or not a particular real-life fact should be recorded as an attribute of a particular entity.

There are three conditions that require such rules: (a) when subtle issues need to be emphasized or the description of an entity, relationship or attribute requires elaboration; (b) when several objects could qualify as one entity, a few associations as one relationship or a number of facts as one attribute, all to varying degrees—the case of fuzzy entities, relationships or attributes; and (c) when an object can be defined either as one entity or another, an association as one relationship or another or a fact as one attribute or another—the borderline case.

4.2.1. Elaboration Rules

Elaboration rules add checking points that can be used at the time of data collection to check whether or not a particular entity, relationship or attribute is established "correctly." Such rules can either state facts that are already known to provide clarifications, consistency checks and to emphasize subtleties; or they may provide a checklist.

Most elaboration rules do not add information, but rather anticipate possible mistakes or misconceptions. An example of a rule for clarification for the entity *license* is: "a license to operate a restaurant must be given by the City's Licensing Department. Licenses issued by licensing departments of other cities should be ignored." Similarly, to avoid inconsistency or mistakes in recording the grade a restaurant has been given by a reviewer, a rule may state that a restaurant can be graded only if it has been reviewed.

Checklists are more informative than general elaboration rules. Consider the entity *reviewer* of restaurants. To provide a clear distinction between persons who simply write restaurant reviews and those who are "truly" reviewers, one could construct a set of rules—for fuzzy entities (see below)—that would set the record straight for each person. Such a task is likely to require much time as it involves speculations about a large variety of instances where persons who write reviews are not restaurant reviewers in the strict sense. If the number of reviewers is relatively small, however, designers may consider a checklist. They can simply list the qualified reviewers in the city—establish the Reviewer List—and formulate the elaboration rule: "a person is a reviewer only if his or her name is on the Reviewer List."

Checklists may take various forms. If the location of a restaurant (i.e. the district or neighborhood) has to be determined, designers can use the Official District Map of a city as a checklist for data collectors to consult when they assign locations to restaurants.

4.2.2. Fuzzy Entities, Relationships and Attributes

4.2.2.1. Fuzzy entities. Rules for fuzzy entities are useful to handle situations when a number of real-life objects could qualify as an entity, and some more so than others. Here a rule must be devised that either expands the definition of the entity or provides additional criteria—general or specific—for establishing the entity.

Consider again the entity *reviewer* as an example. Suppose that a Reviewer List cannot be established, and that the definition of the entity is: "a person who makes his/her reviews of restaurants known to the public." Here we encounter a typical fuzzy case: a great number of people may qualify as reviewers, and some more so than others.

One approach to resolve this problem is to expand the entity's definition and to devise explicit criteria to determine when a review was actually made known to the public. Designers may decide to use a general statement, such as: "a person is a reviewer if his/her reviews appear in a publication that has a circulation of at least 5000." Or, a designer may explicitly state the publications that are "qualified" as sources that make information known to the public: major daily and weekly newspapers and magazines, or books about restaurants.

Another method to resolve a fuzzy case is to provide additional criteria for the selection of a reviewer. A general criterion of that nature may be: "a person qualifies as a reviewer if she or he has published at least 10 restaurant reviews in the last year and if the reviews have been published on a regular basis."

More specific criteria might be given to accommodate special situations. Consider Mr Cook—a retired art critic who is well-known and admired for the occasional restaurant reviews he publishes in a local newspaper. To ensure that his reviews are included in the database, designers may provide yet another instruction: "if a person is well-known as a reviewer, he or she qualifies as a reviewer even if the reviews are published irregularly or in publications with low circulation."

In contrast, one may want to specify that other criteria are of no consequence to the determination of an entity. An example of such a rule is: "a person qualifies as a reviewer regardless of how popular his or her reviews are, whether or not writing restaurant reviews is the main source of income, or of the degree to which this person is knowledgeable about food."

4.2.2.2. Fuzzy relationships. Rules for fuzzy relationships are consulted when it is not clear whether or not an association qualifies as a relationship, or when some associations qualify as a relationship more than others. These rules are likely to constitute one of the largest sets of rules in a database, because they actually define the conditions under which each relationship holds.

Various kinds of problems would require the use of rules for fuzzy relationships.

To exemplify these problems let us consider the relationship "serves" that associates the entities *restaurant* and *type of food.* For this purpose let us also assume that it is a mandatory, many-to-many relationship.

Procedures to establish relationships—Each relationship requires at least one rule that specifies the criteria to use, or the procedure to follow, in order to establish the relationship. For instance, to ascertain whether or not License No. 5428 authorizes the operation of the Minaret restaurant, it is sufficient to check whether or not the name of the restaurant is printed on the license. To determine whether or not this restaurant serves Italian food, however, more complex rules are needed. This relationship requires a rule explaining the procedure by which the type of food is determined, such as: "the type of food that a restaurant serves is determined by the following procedure: each dish on the menu is assigned a type of food; the types of food are then ranked by number of dishes, eliminating types for which less than four dishes are available. The first three on the list are selected as the types of food."

Upper limits on relationships—One-to-many, or many-to-many relationships may require that the number of real-life objects in an entity type corresponding to one real-life object in another be limited. For the "serves" relationship, designers may decide that while a type of food can be served in any number of restaurants, a restaurant can serve up to, say, three distinct types of food.

Relationships that cannot be established—It may happen that a relationship that is mandatory cannot be established for a particular entity. A rule about how to "establish" a relationship in such cases should be devised. A restaurant, for example, may not have a distinct type of food to offer because the number of dishes it serves of each food type is smaller than four. Database designers should anticipate such a case and create a rule similar to: "if there is no evidence that a restaurant serves a specific type of food, assign 'general' as its type of food."

Similarly, one may end up with types of food that are not assigned to any restaurant. The relationship "serves," however, is mandatory, and one must, therefore, create a rule to accommodate such discrepancies. An example of such a rule is: "if a type of food has not been associated with a restaurant, using the procedure of menu examination, assign it to the restaurant which serves the largest number of dishes of that type."

Doubtful cases—A rule should be formulated to guide decisions in case of doubt. Such rules may suggest additional examinations, or they may specify whether or not to give a relationship the benefit of doubt. For example: "in case of doubt whether or not a restaurant serves a particular type of food, assign this food type to the restaurant."

Erroneous information—It may happen that the source of information or the procedure followed to establish a relationship leads to erroneous results. Anticipating such an instance with the relationship "serves," for example, designers may create a rule such as: "even if it is known that some dishes of a certain type of food are not actually served, include this type of food if the score is high enough."

Multiple correspondence—One-to-one, or one-to-many, relationships may present problematic instances where an individual entity which is supposed to relate to only one entity from another entity type actually relates to more than one entity. Therefore, a rule must be created for relationships with these cardinalities that determines whether to split the original entity, or how to select a single entity from among the entities that can possibly relate to it.

The relationship between the entities *restaurant* and *license*—"operates by"—is a relevant example here. Designers must anticipate instances where a restaurant actually has more than one license even though each restaurant is supposed to have only one license. Design specifications and requirements would determine which rule to select. Designers may create a rule that suggests that the restaurant be split into a number of restaurants, each corresponding to a different license. Alternatively, one may advise that if a restaurant has more than one license, the earliest one be recorded.

4.2.2.3. Fuzzy attributes. As illustrated with examples for entities and relationships, rules for fuzzy attributes are important when a number of real-life facts could qualify as one attribute, and some more so than others. Similarly, the types of rules for fuzzy entities and relationships apply to attributes as well. Examples of a few instances of fuzzy attributes are discussed here.

The first example is the use of additional criteria to select the "right" attribute when more than one fact qualifies as an attribute (the *multiple occurrence* case). If, for example, the name of a reviewer is a one-to-one attribute, designers may want to anticipate the unlikely case where a reviewer has more than one name. Here, a rule that specifies additional criteria for determining reviewer name is of help, such as: "if a reviewer has more than one name use the one by which he or she is most commonly known."

Second, a rule for fuzzy attributes may actually dictate a *procedure* to determine a value for the attribute. Consider the question of how to determine the price range of a restaurant. A database designer may prescribe the following procedure: "examine the prices on the menu for dishes that are not appetizers nor desserts. The price range starts with the second lowest price on the list (unless there are at least three dishes available in the lowest price) and concludes with the highest price on the list for which at least two dishes are available."

This procedure reminds us of the third example: a rule that determines how to "establish" an attribute that is mandatory but *cannot be established.* The attribute *price* is mandatory, yet, following the above procedure one may not be able to establish a price range for a restaurant that serves desserts only, or for a restaurant in which the prices change every night according to the daily special. To facilitate data collection in such problematic instances, one may formulate the rule: "if the price of food at a restaurant cannot be determined using the designated procedure, ask the manager to quote the range of price." Note that this rule would probably require the use of a source of information other than the authorized one.

Rules for fuzzy attributes can also guide data collection when the information found in the authorized source is known to be incorrect (the *erroneous information* case). To ensure that a name is entered for each reviewer, for example, a rule can be formulated that states: "if the name found in the authorized source for a reviewer is incorrect, enter it if the correct name is unknown; otherwise, enter the correct name."

Lastly, a one-to-many or many-to-many attribute may require an *upper limit* on the number of values that can be assigned to each entity. One may state, for example, that a restaurant can have no more than three price ranges.

4.2.3. Borderline Cases

Rules for borderline cases are formulated to accommodate instances where a real-life object or fact can be defined either as one entity, or attribute, or another. The rules determine whether objects or facts should be recorded as two distinct entities, or attributes, or whether one entity type, or attribute, is preferred to the other and under what conditions.

An example for a rule that recommends an object to be recorded as two distinct entities or attributes is a possible rule for the name of a reviewer: "if the real name and the pen name of a reviewer are one and the same, record the name for both attributes."

Alternately, the distinction between *address* and *location*, which is the district or neighborhood of a restaurant, may require a rule that specifies conditions for borderline cases. Data collectors may puzzle over whether a certain shopping center should be recorded as a location or as an address. An example of a rule to help them is: "a shopping center that houses more than 20 establishments is considered a location: otherwise it is recorded as an address."

4.3. Rules for domains

Domains define the valid values that each attribute may have. They do not explain, however, how to select the values pertinent to an attribute of a particular entity from among the possible valid values. Rules for a domain are created for that purpose. For instance, while the domain of a restaurant code may be a string of six letters, only a rule can explain how to construct the code for the Minaret restaurant. The most common type of rule here is the general rule and it would be required for the domains of most attributes. In addition, the domains of some attributes would call upon rules for fuzzy domains.

4.3.1. General Rules

These rules give instructions about the construction and selection of values for attributes of individual entities.

Basic rules—The most common type of a general rule is the basic rule which provides the most general instructions about how to select values. Consider the attribute *address* of a restaurant, and suppose the authorized source of information for this attribute is the restaurant's license. An example of a basic rule for the domain of this attribute is: "record the address of a restaurant exactly as it appears on its license."

Content rules—Suppose, however, that designers think that addresses are recorded on licenses inconsistently. While retaining the restaurant license as the authorized source of information, designers may prefer to explicitly state what parts of the address to record, say: number, street, city and zip code. Such a rule is a content rule: it states what information to include in the values for each attribute. Still another form of this rule is to require that at the minimum, these four elements be recorded and further information can be added, if relevant.

General rules are formulated in anticipation of a variety of exceptions and irregularities. Some examples of such irregularities and the rules they require are given below.

Missing-value rules—The purpose of these rules is to instruct about the selection of a value when one knows that a value exists but is unable to find it. For instance, one may not be able to find the real name of a reviewer—an attribute that is mandatory. For some attributes it might be necessary to find the correct value, and designers may explicitly require that the real name be identified. If the database specifications allow for more flexibility, however, designers may formulate a rule that states: "if the real name of a reviewer cannot be ascertained, enter 'unknown'." Here the term "unknown" is a valid value for the attribute. A similar rule may suggest entering a pen name instead, assuming those names are always available.

Sequence rules—Domains for attributes that accept more than one value for a single entity may require a sequence rule: a rule that explains in what order these values be recorded. Consider the attribute *price* of food in a restaurant. If the domain includes all the price ranges charged by a restaurant—for lunches, dinners or a late-night snack—a sequence rule might help to keep data recording consistent.

Shortcut rules—If values for an attribute are relatively long strings of symbols, designers may suggest that some of the values be abbreviated, hopefully with no loss of information.

Quality rules—It may happen that the information taken from the authorized source of information "does not look right," or it does not have an expected quality. For example, the name of a restaurant may not seem like any other name, or the price charged may seem imaginary. To guide a perplexed person who collects the data, designers must formulate a quality rule that explains whether to accept such values, or how to look for the "correct" values.

Authority lists—All the domain rules described here are in the form of a general rule that elaborates on the definition of the domain. Some attributes, however, may require that their domains be explicitly listed. Such lists are called authority lists: they include all the valid values of an attribute. The name of the type of food, is an example of such an attribute. An authority list is needed here so data collectors—and users—know that Middle Eastern, Arabic and Israeli food are very similar to one another and should possibly be recorded under one name.

4.3.2. Fuzzy Domains

Rules for fuzzy domains are required when the data taken from the source of information do not correspond accurately to the domain and its characteristics, or when a domain includes values from other domains. The discrepancy between data found in the real world and the domain as defined by database designers or its values, manifests itself in a variety of instances. Below are several examples.

Lack of correspondence—The data found in the source of information may not correspond to the definition of the domain. Restaurant licenses, for example, may include in their "address" area information that would not usually be considered address information, e.g. the number of the lot on which the restaurant is located, or the number of the original building permit. To resolve instances where the data in a restaurant license do not correspond to the definition of the domain a designer might formulate a rule such as: "omit any information in the 'address' area of the license that is not part of the address."

Incomplete information—Similarly, a rule must be created for instances where the data from the source of information is incomplete. One may instruct, for example: "when the street number is not available, enter '0000', when the street name is not available enter 'STREET'; enter 99999 for unavailable zip codes."

Invalid values—The values found in the source of information may not be valid values for the attribute. Consider the attribute *price* that is charged by a restaurant, where the price is expressed by two numbers separated by a hyphen, for the range of prices that are charged. Anticipating a situation where a restaurant charges only one price for all its meals, like "eat all you can" places, designers should decide whether the price such restaurants charge should be recorded as one number, or as a sequence of two identical numbers separated by a hyphen.

Too many values—The number of values found in the source of information may exceed the permitted limit. For example, even if designers limit the number of telephones a restaurant can have to one, some restaurants may have more than one telephone number. A rule that solves such a discrepancy is: "when a restaurant has more than one telephone number, record the first on their list and add a '+' sign to designate that there are additional telephone numbers."

Indistinguishable values—Two—or more—identical values may need to be distinguished. For instance, it may happen that using a formula to create a restaurant code, two restaurants would be assigned the same code. Anticipating such a mishap, one would devise a rule, in addition to the general rule, that provides instructions about how to construct a code for a restaurant whose code has already been assigned to another restaurant.

Another example of this type of fuzzy domain, even though quite different, is the domain of telephone number. If designers decide to record both the reservations and manager numbers for each restaurant, it is necessary to designate the destiny of each number. Note that a rule to solve such discrepancies would be necessary whenever a divided attribute has only one set of values.

Unreliable data—The data taken from the authorized source of information might be questionable. In such cases a rule should instruct data collection personnel whether or not to adhere to the source and under what circumstances. "Record a restaurant address from the license even if you think it is incorrect (e.g. if it includes typos)," is an example of such a rule.

Values from other domains—Special problems may arise when a domain includes values from other domains. An example of a rule to resolve that type of fuzzy domain is: "if the name of a restaurant is a name of a type of food, record it as the name of the restaurant only if it coincides with the type of food that is served in the restaurant."

4.4. Occurrence

An occurrence rule states whether a relationship or an attribute is mandatory or whether it is optional. These rules are regularly mentioned in the database literature and there is no need to elaborate. One may want to remember, however, that mandatory relationships and attributes require rules for instances where a relationship, an attribute or a domain cannot be established.

4.5. Cardinality

The cardinality of a relationship or an attribute determines whether it is one-to-one, one-to-many or many-to-many. As with occurrence, this concept is well-known and one should only be reminded of the types of rules mentioned earlier that are required for particular cardinalities.

5. DISCUSSION

The typology presented here is based on a set of rules designed to guide data collection for one type of database. While it reflects a variety of types of individual diversity or possible exceptions, this typology is by no means exhaustive: databases of other kinds may necessitate additional types. It is a beginning though, and one that proposes a structure for a more general typology.

The typology identifies discrete irregularities that can be readily recognized in the real world. Yet, some rules would necessarily belong to more than one category. The rule "if the name of a restaurant is a name of a type of food, record it as the name of the restaurant only if it coincides with the food type that is served in the restaurant," for example, resolves more than one type of problem. While it instructs data collection in the case where the domain of the attribute *name* of a restaurant includes a value from the domain *name* of a type of food, it also answers a situation where the name of a particular restaurant is questionable.

Further, the typology would be more useful if general characteristics of entities, relationships and attributes could predict the types of rules necessary for each component. For instance, a rule for indistinguishable values is called upon whenever a divided attribute has only one set of values. Similarly, a rule that puts an upper limit on a relationship is typical of one-to-many and many-to-many relationships and should always be considered for such relationships. Most other rules that are presented here, however, could apply to any component of a database schema. It would help designers if they could determine by the characteristics of a component which rules to consider. It remains the task of future research to discover the relationships between component characteristics and types of rules.

Similarly, further experience and analysis can also point to relationships among the types of rules. A rule for erroneous information (fuzzy relationships and attributes), for example, would require a reconsideration of the rules for authorized sources, and so would a rule for missing-value.

The typology is useful to database designers, however, even in its present form. When constructing the conceptual schema, designers may check each entity, relationship and attribute against each type of rule to determine whether or not a rule is necessary.

For example, when creating the entity *former employer* in a database for an employment agency, one is led to first decide what is the chief source of information about former employers, and which source to use when the chief source is unavailable or when there is more than one source. Next, a designer would consider possible general statements for clarifications or consistency checks. Lastly, the designer would ask which agencies or people may seem to be employers but should not be recognized as such for the purpose of the database. Anticipating these irregularities, he would formulate rules to establish fuzzy entities.

The typology of rules is particularly helpful for developing decision support systems, because such systems provide information in a particular subject area rather than representing processes and procedures. Here, database designers have to anticipate all possible diversity with very little help from the data themselves. The typology can aid, however, the design of databases for other systems as well. Even when a system is designed to replace existing processes and procedures, not all rules to counteract individual diversity are already explicitly stated. City personnel, for instance, may have developed a "sense" for determining the names of a restaurant through experience and they may not recognize that an explicit rule is required. With the typology, designers can uncover *systematically* existing rules and point out situations that require additional rules.

The typology can be expanded, and its structure can be further developed when types of rules used for other databases are integrated. More input would eventually lead to a more general typology that would guide database designers in formulating rules for individual diversity and exceptions. Additional experience may also pave the way to the discovery of the relationships between rule types and between characteristics of schema components and the types of rules they require.

REFERENCES

[1] W. Kent. *Data and Reality: Basic Assumptions in Data Processing Reconsidered.* North-Holland, New York (1978).
[2] J. Martin and C. Mcclure. *Structured Techniques for Computing.* Prentice-Hall, Englewood Cliffs, New Jersey (1985).
[3] S. Cerri (Ed.). *Methodology and Tools for Data Base Design.* North-Holland, New York (1983).
[4] R. Fidel. *Database Design for Information Retrieval: A Conceptual Approach.* Wiley, New York (1987).
[5] P. P. S. Chen. The entity-relationship model—toward a unified view of data. *ACM Trans Database Systems* **1**, 9–36 (1976).
[6] *Anglo-American Cataloguing Rules*, Second Edition (Edited by M. Gorman and P. W. Winkler). American Library Association, Chicago (1978).