2

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

THE PROBLEM OF UNIQUE NAME VIOLATIONS
IN DATABASE INTEGRATION

by

Renae M. Beyer

March, 1991

Thesis Advisor: H.K. Bhargava

Approved for public release; distribution is unlimited

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED		1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE		
4. PERFORMING ORGANIZATION REPORT NUMBER(S)		5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School	6b. OFFICE SYMBOL *(If applicable)* 37	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS *(City, State, and ZIP Code)* Monterey, CA 93943-5000		7b. ADDRESS *(City, State, and ZIP Code)* Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION	8b. OFFICE SYMBOL *(If applicable)*	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS *(City, State, and ZIP Code)*		10. SOURCE OF FUNDING NUMBERS

Program Element No	Project No	Task No.	Work Unit Accession Number

11. TITLE *(Include Security Classification)*
THE PROBLEM OF UNIQUE NAME VIOLATIONS IN DATABASE INTEGRATION

12. PERSONAL AUTHOR(S) Beyer, Renae M.

13a. TYPE OF REPORT Master's Thesis	13b. TIME COVERED From To	14. DATE OF REPORT *(year, month, day)* March 1991	15. PAGE COUNT ~~145~~ 146

16. SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17. COSATI CODES

FIELD	GROUP	SUBGROUP

18. SUBJECT TERMS *(continue on reverse if necessary and identify by block number)*
Synonym, Homonym, Database Integration, Unique Name Violations, Naming Conflicts, Quiddity

19. ABSTRACT *(continue on reverse if necessary and identify by block number)*

When multiple database schemas are integrated, there are often conflicts in the naming of attributes within schemas. These conflicts must be detected and resolved prior to successful integration of the schemas. This thesis describes a method for automatically detecting such naming conflicts, which adapts and enhances a method for detecting similar conflicts in (mathematical) model integration. The method relies on the representation of semantic information, not found in data dictionaries, about the data elements or attributes present in the various schemas. The information about data elements is then used by mechanical inference procedures to automatically determine whether two distinctly named elements in fact represent the same object (the synonym problem), or if data elements with the same name in different schemas actually represent different objects (the homonym problem). The expected accuracy and errors of these procedures, and results obtained from a set of experiments on the use of this method, are also presented.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS REPORT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL H. K. Bhargava	22b. TELEPHONE *(Include Area code)* (408) 646-2264	22c. OFFICE SYMBOL AS/Bh

83 APR edition may be used until exhausted
All other editions are obsolete

The Problem of Unique Name Violations in Database Integration

by

Renae M. Beyer
Major, United States Army
B.S., Kearney State College

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL
March 1991

Author: Renae M. Beyer
Renae M. Beyer

Approved by: H. K. Bhargava
Hemant K. Bhargava, Thesis Advisor

Daniel R. Dolk
Daniel R. Dolk, Second Reader

David R. Whipple, Jr., Chairman
Department of Administrative Sciences

ABSTRACT

When multiple database schemas are integrated, there are often conflicts in the naming of attributes within the schemas. These conflicts must be detected and resolved prior to successful integration of the schemas. This thesis describes a method for automatically detecting such naming conflicts, which adapts and enhances a method for detecting similar conflicts in (mathematical) model integration. The method relies on the representation of semantic information, not found in data dictionaries, about the data elements or attributes present in the various schemas. This information about data elements is then used by mechanical inference procedures to automatically determine whether two distinctly named elements in fact represent the same object (the synonym problem), or if data elements with the same name in different schemas actually represent different objects (the homonym problem). The expected accuracy and errors of these procedures, and results obtained from a set of experiments on the use of this method, are also presented.

A-1

TABLE OF CONTENTS

I. INTRODUCTION

This thesis examines and develops a method for automatically detecting possible naming problems of data elements prior to database integration. These naming problems or conflicts involve synonyms and homonyms. Synonyms are data elements in two or more different databases which are given different names but contain information about the same thing. Homonyms are data elements in two or more different databases which are given the same names but contain information about different things. For example, one database might call the data element which contains a person's **given name**, "FIRSTNAME," while another calls it "FNAME." Conversely, that same database might call the data element which contains the **last name** of an individual, "NAME," while another calls the data element which contains the **full name** of an individual, "NAME."

These naming problems have been identified in database literature and it is accepted that prior to schema integration, naming problems must be detected and resolved (e.g., Bhargava, Kimbrough, and Krishnan 1990; Kamel and Hsiao 1990; Batini, Lenzerini, and Navathe 1986; Wang and Madnick 1989; Larson, Navathe, and Elmasri 1989; and Hayne and Ram 1990). Further, several methodologies have been proposed for detection (e.g., Batini et al. 1986; Larson et al. 1989; Mannino and Effelsberg 1984), but are not supported with automated tools. The methodologies require the database designer or administrator to detect the problems by systematically examining the data elements in each database. The database designer or administrator are able to locate many of these problems by reviewing the data dictionaries and additional information about the databases from other sources (e.g., users). As noted, these methods can be tedious and extremely time-consuming.

More recently, a method for supporting **automatic detection** of possible naming problems in model integration has been proposed by Bhargava, Kimbrough, and Krishnan, hereafter referred to as Bhargava et al. (1990). This method requires a database designer to further define each model variable by providing dimensional information and information (called "quiddity"), see Chapter III, about the nature or essence of the data contained in the variable. This thesis addresses the applicability of quiddity in detecting naming conflicts prior to database integration.

A. BACKGROUND

A database is simply a computerized record-keeping system. The information in a database is stored (at the lowest level) in units called data elements or data items. Each data element has a unique name associated with it. For example, the data element which contains an individual's social security number could be called "SSN." Data elements also have other assigned characteristics such as type and size. The *type* tells whether the data element is alphabetic, numeric, or a special character. The *size* describes the length of the data element, e.g., the number of characters that fit in the field. All this information about data, including relationships between the data elements, comprise the database *schema*. The schema provides a complete and logical view of the database. (Date 1990)

With the proliferation of database technology, many organizations need to access and share information between databases to facilitate decision making, operations planning and control, and strategic planning.

> This situation has led to the emergence of the *heterogeneous distributed database* scenario. In this scenario, a variety of large and small computers, each with its own autonomous and often incompatible DBMS [Database Management System], may be tied together in a network. This network could consist of local area, wide area, and long-haul networks. Under current technology, however, a user accessing any database in this network must abide by the syntactic and semantic rules of that database (Cardenas 1985). ... A true heterogeneous DBMS should support an environment in which any user in the network is given an integrated and tailored view, while in fact the data could physically reside on a

single or several databases managed by different and possible heterogeneous DBMS. This level of data access and sharing is known as database *integration*. (Kamel et al. 1990)

One of the steps to be performed before database integration can occur is schema integration, i.e., the integration of the local schemas of the databases involved into one global schema. Several different problems are encountered during schema integration. The one we are primarily concerned with involves the different ways similar information is captured in the databases being integrated. The fact that the same data may be described differently in each local schema presents some very challenging issues. Zviran and Kamel (1989) classify these issues into four general areas. They are:

1. **Name Conflicts**. These conflicts exist when there are synonyms, i.e., data elements with the same name but representing different concepts, and homonyms, i.e., data elements with different names but representing the same concept. For example, one database might call the data element which contains a social security number, "SSN," while another calls it "SSNO" (synonym). Or, two databases might both have data elements called "DATE" but in the first database the date represents the current date while in the other it represents an individual's employment date (homonym).

2. **Structural Conflicts**. These conflicts exist when the same information is represented in different structures in each schema. For example, an individual's full name (first, middle, and last) is maintained as a single data element in one database but is split into three data elements in another.

3. **Scale Conflicts**. These conflicts exist when the same facts are expressed in different units of measure in each schema. For example, a person's height may be captured in inches in one schema and feet in another.

4. **Conflicts in Application Semantics**. These conflicts exist when perceptions about information differ between schemas. For example, the relationship between two objects in a schema is represented as "one to many," but is represented in another schema as "one to one."

Identifying and resolving these conflict issues is a critical step in successful schema integration.

B. PROBLEM DESCRIPTION

While technological improvements have kept pace with the increased requirements for exchange and sharing of information, automated tools or methods to facilitate the physical integration of the data prior to exchange, i.e., schema integration, have been slow in coming. For example, those knowledgeable in the process of database integration continually emphasize the importance of identifying naming conflicts among databases prior to integration but fail to provide tools with which to accomplish this crucial yet painstaking task (Bhargava et al. 1990).

The problem of *naming conflicts*, i.e., the violation of the unique names assumption[1] (UNV) in database integration occurs when there are synonyms or homonyms among the data elements. For example, one database might call the data element which contains a **social security number**, "SSN," while another calls it "SSNO" (synonym) or that same database might call the data element which contains the **last name** of an individual, "NAME," while another calls the data element which contains the **full name** of an individual, "NAME" (homonym). Another interesting twist to the problem is that it is possible to have data elements with different names containing information about the same thing but having different values. This can happen when there are small measurement errors. For example, two databases with data elements called "HEIGHT" and "HT," respectively, capturing information about the "height" of the **same** person could have different values, e.g., 68" versus 67". If the same data element name had been used, the problem of two different values would be easily detected. Not resolving these and similar conflicts before integrating would result in a database which clearly has redundant data (e.g., two data elements containing social security numbers) and would in all likelihood develop serious consistency problems (e.g., similar fields with different values).

How are these conflicts detected? There are two basic methods currently used in identifying these conflicts. In the first method, the data element names are compared syntactically, and data types (e.g., numeric, alphabetic) and field lengths are matched. The second method involves an

[1] "That every individual has at most one name, unless stated otherwise, is often a useful and convenient assumption in software systems, and is called *the unique name assumption*." (Bhargava et al. 1990)

examination of the data dictionary. The data dictionary has more descriptive, semantic information about the data elements. However, it is written in non-formal, human language, which is not amenable to machine inference.

How **can** we identify these conflicts through automation? Clearly, we need more semantic information: **information about what the data element represents**. Bhargava et al.'s (1990) method for supporting automatic detection of possible naming problems requires that each data element be further defined in terms of dimensional information and information about the nature or essence (**quiddity**) of the data contained in the data element. The quiddity of a data element is specified using various rules of formulation and a given vocabulary. The objective of this approach is to identify pairs of data elements with *possible* unique names violations by comparing the dimensional information and quiddity of each data element in the databases being integrated. The premise of this approach is that if two data elements have the same quiddity, it is fairly likely that they refer to the same concept. This automated approach will not specifically identify naming conflicts, rather, it will result in a list of *possible* conflicts. Human interaction is required to confirm specific conflicts. The intent is to develop a list of possible conflicts which comes as close as possible to the "correct list."

It is fairly straightforward to provide dimensional information for each data element because there are a finite set of dimensions (e.g., length, mass, time, volume). However, the quiddity of each data element is more complex to define. Quiddity must be stated in a well defined form, a type of *formal language*, in order to be read and compared by a computer. For example, the quiddity of the data element "NAME" (referring to the <u>last name</u> of an individual) discussed above, would be "**last(name(person))**."[2]

[2] The representation of *quiddity* is discussed in greater detail in Chapter III.

C. THESIS OBJECTIVES

The aim of this thesis is to examine several aspects of quiddity, broadly classified into those dealing with quiddity concept definition, quiddity acquisition, and quiddity manipulation and inferencing procedures.[3] Specifically, the following questions in each area will be addressed.

1. Quiddity Concept Definition

Is the idea of quiddity, as defined by Bhargava el al., a practical tool for use in the integration of databases? Specifically, is the concept of quiddity sufficiently rich or expressive in the database context? If not, in what ways can the concept be modified to one that is rich enough? Can quiddities provide a basis for automatically detecting unique name violations, or will the quiddities create more problems than they solve?

2. Quiddity Acquisition

Can this method be easily understood and applied by database designers? In other words, will two individuals always develop the same or equivalent quiddity definition given identical data elements, information, and training? If not, how can the acquisition process be supported?

3. Quiddity Manipulation and Inferencing Procedures

What kinds of inference procedures can be defined to utilize this quiddity information in order to automatically detect naming conflicts? How can these procedures be implemented? What is the accuracy and error rate of these procedures, in terms of Type I and Type II errors?[4]

[3] Bhargava et al. (1990) have discussed a formal, functional representation for quiddities. For our purposes, a less formal tabular notation will suffice. Hence, this thesis is not concerned with issues in quiddity representation.

[4] A Type I error is indicating a naming problem when there is none. A Type II error is failing to indicate a naming problem when there is one.

D. METHODOLOGY

The research for this thesis follows these steps:

1. Conduct preliminary experiment. In this experiment, explain the concept of quiddity to a group of six Computer Systems Management students and ask them to then develop "quiddities" for a sample set of data elements in a database. This first experiment will be primarily used to ensure that all subjects understand the concept and what is being asked of them, in other words, to eliminate any "noise" which could interfere with the analysis of the concept itself.

2. Refine, analyze, and enhance the concept based on results of preliminary experiment. Provide feedback to students on "correctness" of their experiment answers. Develop and present several procedures for comparing the quiddities in the experiment.

3. Conduct primary experiment with the same individuals who participate in the first experiment, using a new sample set of data elements. Present any new rules or instructions in developing quiddity to the students based on the analysis and any enhancements developed in step 2.

4. Analyze the results of the primary experiment by applying the comparison procedures developed in step 2 to the students' quiddity definitions.

5. Evaluate results of primary experiment, discussing any shortcomings in the quiddity concept or inference procedures. Discuss future areas of research.

E. THESIS STRUCTURE

Our research is presented in five chapters. Chapter II provides a general review of related work in detecting naming conflicts in database integration. Several issues related to our proposed method for UNV detection are addressed in Chapter III. Section A presents a detailed description of this proposed method. Results of a preliminary experiment along with a refined concept based on the experiment analysis appear in Section B. Finally, Section C discusses several quiddity manipulation and inferencing procedures. Chapter IV describes the primary experiment and presents detailed experiment results and analyses. Chapter V presents our conclusions and suggests issues for future research.

II. REVIEW OF RELATED WORK

Our aim in this chapter is to present a general overview of current literature pertaining to database integration, with emphasis on those which address methods for the detection of naming conflicts or present automated tools for use in detecting such conflicts.

A. SCHEMA INTEGRATION

The literature to date views schema integration in two contexts. The first, commonly referred to as *view integration*, generates a global conceptual description or logical integrated schema of a proposed database during database design. The second, referred to as *database integration*, generates the global schema of a group of databases in distributed database management. (Batini et al. 1986)

Kamel et al. (1990) have reviewed and grouped current literature into the context of view integration and database integration. Previous research has focused on schema integration in the context of view integration (Batini, Lenzerini, and Moscarini 1983; Elmasri and Navathe 1984; Elmasri and Wiederhold 1979; Motro and Buneman 1981; Navathe, Sashidhar, Elmasri 1984; and Sheth, Larson, Cornellio, and Navathe 1987), while some have addressed issues of schema integration in the context of database integration (Kamel et al. 1990; Dayal and Hwang 1984; Deen, Amin, and Taylor 1987; DeMichiel 1989; and Wang and Madnick 1989). Batini et al. (1986) have also provided a general survey on view integration methodologies.

Schema integration, regardless of context, involves many complex issues. One of these issues is conflict identification and resolution, specifically, conflicts in name or unique names violations. Although methodologies do address this issue, Bhargava et al. (1990) state that most methods assume that unique names violations are dealt with prior to integration (Casonova and

Vidal 1983; Yao, Waddle and Housel 1982). Others have suggested that naming conflicts can be easily handled simply by renaming (Dayal and Hwang 1984), but have not proposed how to handle the conflicts.

B. APPROACHES ADDRESSING NAMING CONFLICTS

Larson et al. (1989) propose a method of schema integration which provides assistance in the detection of naming conflicts. (This methodology builds on previous works (Elmasri and Navathe 1984; Navathe, Sashidhar, and Elmasri 1984; and Elmasri, Larson, and Navathe 1986.)) Basically, this method involves the application of certain criteria to attributes (data elements) in order to determine "attribute equivalence." Equivalent attributes have several characteristics in common and can be integrated. Examples of attribute characteristics considered are uniqueness, cardinality, domain, static and dynamic semantic integrity constraints, security constraints, allowable operations, scale, and others that a database administrator feels are important. Then, based on certain equivalence properties, the attributes are integrated. This concept is also used to define object and relationship set equivalences for integration purposes. The criteria for attribute equivalence is applied to naming conflicts which can then be identified and resolved. However, this is a tedious, manual process.

Mannino and Effelsberg (1984) have suggested an integration process using assertions, made by database designers, about semantic equivalence between objects. While very similar to Larson et al. (1989) above, this methodology is not as detailed in its treatment of equivalence. Here again, naming conflicts are found through a manual process.

C. AUTOMATED TOOLS FOR SCHEMA INTEGRATION

Larson et al. (1989) have designed and implemented a schema integration tool based partially on their concept described in Section B. With this tool, the database administrator is shown descriptions of the schemas being integrated. The database administrator then specifies all equivalences between schema objects and interactively integrates the schema. While this is a step toward automating the process, the database administrator must still "manually" establish all equivalence characteristics before the schema is "automatically" integrated.

Hayne and Ram (1990) have developed a knowledge based system called MUVIS (Multi-User View Integration System) to support the design of distributed object-oriented databases. This system automates the view integration process as proposed by Navathe et al. (1986). MUVIS aids designers in modeling user views using the Extended Entity Relationship Model and integrating these views into a global conceptual view. MUVIS's expert system compares objects and computes equivalence assertions about these objects using heuristics. Integration rules are then applied and the designer confirms the integration. The designer determines whether there is a naming conflict prior to integrating when he or she confirms the integration.

Hayne and Ram (1990) also reviewed other design tools that are currently available. Several design tools for view modeling and integration have been implemented using the expert system approach. These systems (Bouzeghoub, Gardarin, and Metais 1985; Choobineh, Mannino, Nunamaker, and Konsynski 1988; and Dogac, Yuruten, and Spaccapietra 1989) do not provide graphical interfaces but do allow the specification of incomplete designs and can justify and explain results produced. Again, these tools may automate part of the integration process but do not automate the actual detection of naming conflicts. These conflicts are found through interaction between the designer and the tool.

III. USE OF QUIDDITIES IN AUTOMATIC DETECTION OF NAMING PROBLEMS

Our aim in this chapter is to describe in detail a proposed method wherein quiddities of data elements are declared and used in the automatic detection of naming conflicts. This idea was first developed by Bhargava et al. (1990), and we present a summary of their approach in Section A. Section B presents the results of a preliminary experiment conducted to provide initial data about the applicability of the method in detecting naming conflicts. These results, a deeper analysis, and a linguistic perspective are employed to propose refinements to the method. Section C proposes quiddity manipulation and inferencing procedures to be used by the automated process.

A. CONCEPT AND MOTIVATION

Bhargava, Kimbrough, and Krishnan (1990) have proposed a method for supporting automatic detection of possible naming problems, specifically, unique names violations in model integration.[5] Their contention is that in order for any automated system to recognize that two variables with different names represent the same information, or vice versa, a system requires more information **about** these variables. This method attempts to develop a principled means of providing and expressing that information. It requires capturing two categories of information about each variable, its dimension and quiddity. The premise is that *if two syntactically distinct variables have the same or equivalent dimension and quiddity, a* ***possible*** *unique names violation is indicated.* (Bhargava et al. 1990)

[5] All quotes in this chapter (unless otherwise noted) have been borrowed from Bhargava et al. 1990.

1. Representing Dimensional Information

The task of identifying dimensional information for each data element is simple because there is a small number of dimensions (e.g., length, mass, time, volume) recognized in most applications. Even if other dimensions such as currency are added, the set has few elements. Additionally, there is a "place holder" (represented by 1) for dimensionless quantities, such as percentages. Derived dimensions (e.g., volume, acceleration, weight, power) are also allowed.

For two reasons, Bhargava et al. suggest the use of abstract dimensional expressions, e.g., *currency* rather than *dollars*, even though dimensional information is best captured using three components: dimension,[6] unit, and scale. First, the unit information of "dollars" can be captured by the dimensional component, "currency." Second, the use of the most abstract dimensional expression reduces Type II errors when discovering naming problems (Bhargava et al. 1990). For example, suppose a variable is used in two models to measure a quantity of apples. In the first model, the variable X (for apples) is measured in **bushels**. In the second model, the variable Y (also for apples) is measured in **quarts**. There is a unique names violation, but the rules will not find it because the dimensions are not the same. Since both bushels and quarts are measures of volume, the dimension could be stated more generally as "volume," causing the naming violation to be detected.

2. Representing Quiddity

The task of defining the **meaning** or the quiddity[7] of each variable is more complex. The quiddity of a data element provides a description of what the data element is about. Clearly, quiddities must be stated unambiguously, in a *formal language*, in order that they be readable and

[6] "Some authors use the term *quantity* ... [in place of] ... the term dimension ... [as it is used here]." (Bhargava et al. 1990)

[7] "From the *Oxford English Dictionary*, quiddity is 'The real nature or essence of a thing; that which makes a thing what it is.' Of course, ... [the method's] language for expressing quiddities is only a model, or approximation, of genuine quiddity, if it exists." (Bhargava et al. 1990)

comparable by a computer. Bhargava et al. establish five categories for capturing the quiddity of a variable: stuff, types of stuff, attributes of stuff, types of attributes of stuff, and metafunctions. To specify valid quiddity expressions, a basic vocabulary for each of the five components is provided.[8] To develop the quiddity of a variable, each of the five components (described below) are examined and, if applicable, declared. The example shown in Figure 1 is designed to illustrate this definition process for each of the components.

> What is the quiddity for this variable?
>
> - **Variable Name:** *purchase_cost*
> - **Data Dictionary Description:**
>
> "Average cost of purchasing a Dodge truck during the month of July."

Figure 1 Illustration Variable

a. *Components of Quiddity*

(1) Stuff. Stuff answers the question "**what is the variable about**?" *Stuff* is usually indicated by a noun, describing individual things or collections of individual things, such as cars, trucks, or ships. What is the variable, shown in Figure 1, about? It's about a **truck**. Therefore, **truck** is the *stuff* component of this variable's quiddity.

Additionally, a *stuff* term may have *arity*[9] if one or more arguments are required to fully define the *stuff* term. With quiddity, arguments are added to the definition, when necessary, to further define *stuff*. Suppose "path" is the *stuff* expression. In this case, we

[8] For the purposes of the following discussion and examples, assume all terms used in developing quiddity are a part of an established basic vocabulary.

[9] *Arity* identifies the number of arguments required to specify a function. For example, the function of "addition" has an "arity of 2" because you must have two arguments in order to perform the function, in other words, to add. Division also has an "arity of 2," whereas the square root function has an "arity of 1" (you only need one argument to find the square root).

would need to know the two end points of the path in order to define the exact path. Thus, "path" has an arity of 2 since it has two arguments (the two end points). There is no limit for the *arity* of a *stuff* term except that it be finite. Of course, some *stuff* expressions need no arguments (e.g., apple or ship) and have an arity of 0.

In our example, we are interested in a truck purchased **during a given month**, July. In this instance, the *stuff* expression, **truck**, should be further defined because we are concerned with the **truck** at a <u>specific point in time</u>. Therefore, **truck** has arity of 1, with the *argument* **month**.

(2) Stuff Type. Stuff type answers the question "**what <u>sort of</u> or <u>kind of</u> *stuff* is it**? *Stuff types* further describe *stuff*. For example, with both *stuff* and *stuff type* we can distinguish between a "truck tire" and a "tire truck." In the first case, what is the variable about? It is about a tire. What sort of tire? A truck tire. Thus the *stuff* is **tire** and the *stuff type* is **truck**. However, in the second case, the variable is about a truck. What sort of truck? A tire truck. Thus the *stuff* is **truck** and the *stuff type* is **tire**. (Bhargava et al. 1990) To continue with the example in Figure 1, the *stuff type* of **truck** (*stuff*) is **Dodge**.

(3) Stuff Attribute. Stuff attributes answer the question "**what is it about <u>the *stuff*</u> that you are interested in**? *Stuff attributes* represent information about some aspect of the *stuff* we are interested in. From the example above, what is it about a **truck** that we are interested in? The cost. Therefore, **cost** is the *stuff attribute* of the variable **purchase_cost**.

(4) Stuff Attribute Type. Stuff attribute types answer the question "**what <u>sort of</u> or <u>kind of</u> *stuff attribute* is it**? From above, the *stuff attribute* was **cost**. What sort of **cost** are we interested in? **Purchase** cost. Thus, **purchase** is the *stuff attribute type* qualifying the *stuff attribute* **cost**.

(5) Metafunctions. "*Metafunctions* capture information about the variable associated with the quiddity." Examples of *metafunctions* are average, maximum, minimum, sum, and variance. Type II errors in indicating possible naming violations can often be reduced by identifying *metafunctions* in quiddities. For instance, if *A* and *B* are variables for the price of fuel, but *A* is an average price while *B* is not, then no unique names violation should be indicated (Bhargava et al. 1990). From our illustration example, the *metafunction* associated with the variable *purchase_cost* is **average.**

b. Formal Representation

For a computer to be able to compare the quiddities of variables, the quiddities must be represented in a standard format or *formal language.* Bhargava et al. (1990) recommend and develop a rigorous representation in a *formal language*, for capturing quiddity information. This representation is illustrated using the *purchase_cost* variable in Figure 2. In this representation, there may be instances where there are multiple terms in a component. When this happens, the terms are listed alphabetically, to remove ambiguity. Three additional examples of this representation are provided. While these examples are somewhat contrived and simplistic, they demonstrate the basic steps taken in developing the quiddity for a variable.

Quiddity Representation

Metafunction(Stuff Attribute Type(Stuff Attribute(Stuff Type(Stuff(Arg 1, ..., Arg *n*))))

Quiddity of *purchase_cost*: average(purchase(cost(Dodge(truck(month)))))

Figure 2 Quiddity Representation

(1) Example 1. Consider a variable which captures information about the status of an unmanned fighter aircraft. What is the variable about? An **aircraft** (*stuff*). Does **aircraft** need further definition, or, in other words, is aircraft a function of something else? No,

so **aircraft** has an arity of 0 (no *arity arguments*). What sort of **aircraft** (*stuff*) is it? It is an **unmanned** aircraft. It is also a **fighter** aircraft. We have two *stuff types*. What is it about the **aircraft** that we are interested in? Its **tail number**[10] (*stuff attribute*). What sort of **tail number** is it? We have no further information so we do not have a *stuff attribute type*. The quiddity representation for this example is shown in Figure 3.

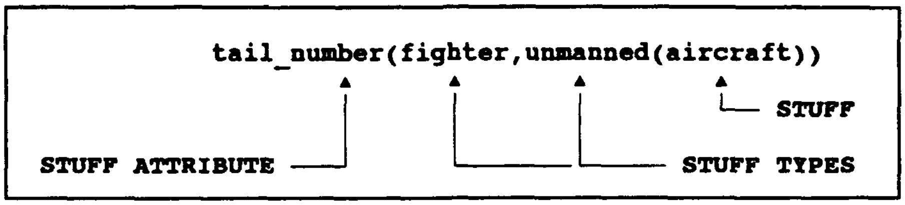


Figure 3 Quiddity Representation -- Example 1

(2) Example 2. Consider a variable which captures information about the retail cost of an IBM personal computer. What is the data element about? A **personal computer** (*stuff*). Do we need any arguments to further define **personal computer**? No, thus there are no arity arguments (arity 0). What sort of **personal computer** (*stuff*) is it? It is an **IBM** (*stuff type*). What is it about the **personal computer** that we are interested in? The **cost** (*stuff attribute*). What sort of **cost** is it? **Retail** (*stuff attribute type*) cost. The quiddity representation is shown in Figure 4.

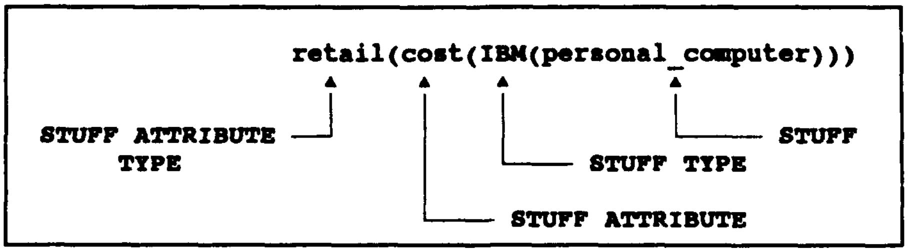


Figure 4 Quiddity Representation -- Example 2

[10] Since *tail number* is a word phrase denoting one concept, the formal quiddity representation connects the words in this form "*tail_number*." This representation allows us to distinguish between word phrases which denote one value for a component (e.g., *tail_number* for the *stuff attribute* component) and two distinct values for a component (e.g., *unmanned* and *fighter* for the *stuff type* component).

(3) Example 3. Consider a variable which captures information about the current replacement cost of a foreign car. What is the variable about? A **car** (*stuff*). Does **car** need any arguments to define it or, in other words, is it a function of something? Yes, we are interested in a **car** at a specific point in **time**. Therefore, **car** has an "arity of 1" with the *argument* **time**. What sort of **car** (*stuff*) is it? It is a **foreign** (*stuff type*) car. What is it about the **car** that we are interested in? The **cost** (*stuff attribute*). What sort of **cost** is it? **Replacement** (*stuff attribute type*) cost. The quiddity representation is shown in Figure 5.

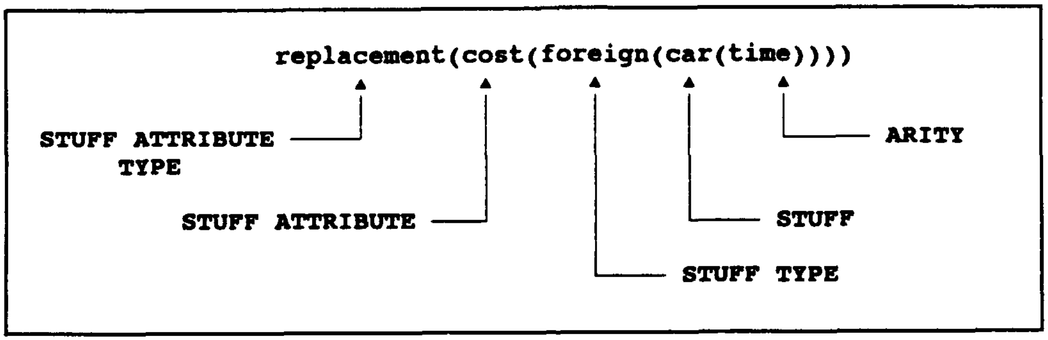


Figure 5 Quiddity Representation -- Example 3

c. *Validity Rules*

Given a basic vocabulary for each category, the following rules for determining valid stuff terms apply (Bhargava et al. 1990, 15).

1. If α is in the vocabulary of basic stuff expressions, then α is a valid stuff term, providing that each of its arguments has the form *arg(n)*, where *n* is an integer identified with a declared variable (or is a declared variable-indicating expression).

2. If α is in the vocabulary of basic stuff expressions, then α[*arg(n)*] is a valid stuff term, where α[*arg(n)*] has one more argument than α and *n* is an integer identified with a declared variable (or is a declared variable-indicating expression) with a quiddity of *index*.

3. $\phi(\alpha)$ is a valid stuff term if α is a valid stuff term and ϕ is in the vocabulary of stuff types.

4. $\phi(\alpha)$ is a valid stuff term if α is a valid stuff term and ϕ is in the vocabulary of metafunctions.

5. Nothing else is a valid stuff term.

Given the above rules for determining valid stuff terms, the following rules for determining valid quiddity terms apply (Bhargava et al. 1990, 16).

1. If α is a valid stuff term, then α is a valid quiddity term.
2. $\phi(\alpha)$ is a valid quiddity term if α is a valid stuff term, and ϕ is in the vocabulary of stuff attributes.
3. $\phi(\alpha)$ is a valid quiddity term if α is a valid quiddity term and ϕ is in the vocabulary of metafunctions.
4. $\phi(\alpha)$ is a valid quiddity term if α is a valid quiddity term and ϕ is in the vocabulary of stuff attribute types.
5. $\alpha \bullet ß$ and α / ß are valid quiddity terms if α and ß are valid quiddity terms.
6. Nothing else is a valid quiddity term.

3. General Observations

a. *Quiddity Component Definitions*

The *stuff* term must be correctly identified in order to accurately capture quiddity because all other quiddity components are built upon the *stuff* term. If this term is accurately defined, the other components are determined with relative ease. However, it is confusing and often difficult to correctly determine the *stuff* term.

Recall the variable *purchase_cost* which captures information about the cost of purchasing a Dodge truck in the month of July. Originally, we answered the questions as follows. What is this variable about? It's about a **truck** (*stuff*). In this case, the purchase cost of the **truck** is a function of **month**, therefore the *arity argument* is **month**. What sort of **truck**? A **Dodge** (*stuff type)* truck. What is it about the **truck** we are interested in? The **cost** (*stuff attribute*). What sort of **cost** is it? **Purchase** (*stuff attribute type*) cost. There is a *metafunction* of **average**. The quiddity representation is shown again in Figure 6, Example (a).

What would happen if we identified a different *stuff* term?[11] What is this variable about? It's about **purchasing** a truck in **July**. Therefore, the *stuff* term is **purchase** and the *arity argument* is **month**. What sort of **purchase** is it? It is a **truck** (*stuff type*) purchase. What sort of **truck**? A **Dodge** (*stuff type*) truck. What is it about the **purchase** that we are interested in? The **cost** (*stuff attribute*). What sort of **cost**? We have no further information so there is no *stuff attribute type*. There is a *metafunction* of **average**. The quiddity representation for this set of questions is shown in Figure 6, Example (b).

Although the examples provided are somewhat contrived, they do demonstrate that the categories of quiddity appear sufficient to capture the **meaning** of the variables' data. Yet the method used to determine the component definitions is not structured enough to elicit the same answer from different people.

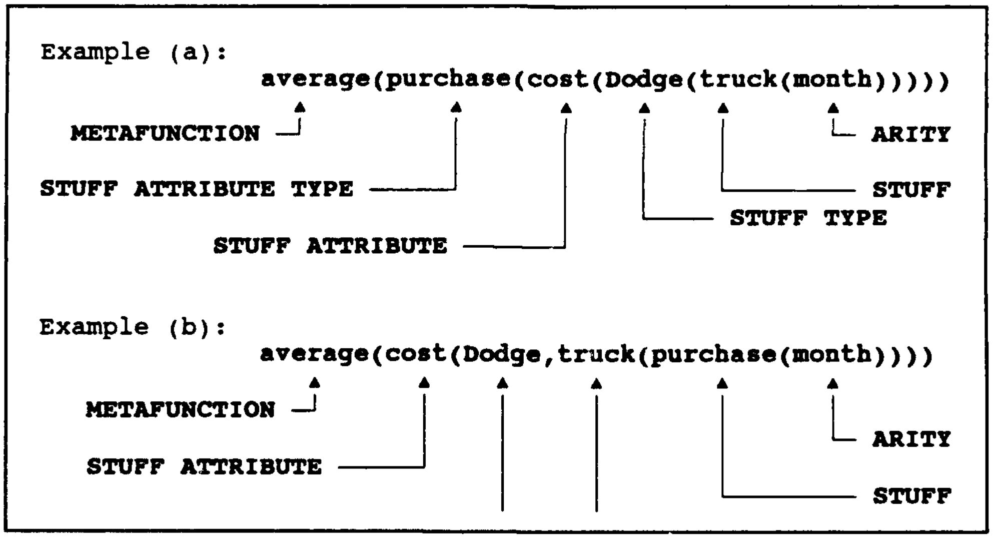


Figure 6 Two Examples of Quiddity Representation for the Variable *Purchase_Cost*

[11] It is recognized that there would be a set vocabulary available for choosing these terms. However, a particular word or words can be applied in more than one quiddity category depending upon need, as shown in paragraph A.2.a.(2), where both **truck** and **tire** are stuff and stuff types.

b. Quiddity Equivalence Rules

An important aspect of the authors' premise, i.e., *if two syntactically distinct variables have the same or equivalent dimension and quiddity, a possible unique name violation is indicated*, is the notion of quiddity equivalence. What exactly constitutes quiddity equivalence? One issue addressed in the paper was whether the order of the quiddity components in the representation is important in establishing equivalence. For example, does it matter if the representation is *stuff attribute type(stuff attribute(stuff type(stuff)))* or *stuff attribute(stuff attribute type(stuff type(stuff)))*? No conclusion, one way or the other was presented. However, the authors did state that this ambiguity could be reduced by stipulating validity conditions of quiddity expressions and introducing equivalence transforms. The quiddity validity conditions were discussed earlier. An example of an equivalence transform would be to state that *stuff attribute(stuff) = stuff(stuff attribute)*. However, in the implementation, only the most straight forward pattern matching rule was used.

There are other aspects of equivalence which were not specifically addressed. Does it make a difference which category a term falls within as long as the term is included in the quiddity expression? For example, in Figure 6, are the two quiddities depicted equivalent? The same words are in each description! Are terms which are synonyms equivalent, i.e., are **cost** and **price** equivalent? These issues are discussed further in following sections.

B. QUIDDITY ACQUISITION

To summarize, Bhargava et al. proposed that each model variable be further defined in terms of its dimension and quiddity. A UNV is indicated if, and only if, both the dimension and the quiddity of two variables are equivalent. If the dimensions are not equivalent, it follows that the variables do not represent the same information and no UNV should be detected. Similarly, if the quiddities are not equivalent, it again follows that the variables do not represent the same information and no UNV should be detected. Since this thesis focuses primarily on the feasibility of quiddity as it applies to detecting naming problems during database integration, the dimension aspect will be ignored in further discussions. It would be unnecessary to examine equivalence of quiddities of two variables if the dimensions are not equivalent. Thus, when checking for quiddity equivalence between data elements in our experiments, we will assume that their corresponding dimensions are equivalent.

1. Preliminary Experiment

The purpose of the preliminary experiment was to provide initial data about the acquisition of quiddity. Additionally, this experiment was intended to eliminate "noise" in the primary experiment thus preventing interference with the analysis of the concept itself. In other words, we wanted to ensure that all subjects had a clear understanding of the concept before conducting the primary experiment.

a. Subjects

Six Naval Postgraduate students enrolled in the Computer Systems Management (CSM) Curriculum participated in the experiment. The students were randomly selected and had varying military backgrounds; Army (2), Navy (3), and Marine Corps (1). All had varying degrees of "computer expertise," from little or none when beginning the CSM Curriculum to having an undergraduate degree in Computer Science. All students had completed a course in the application of database management systems, so all had a common background in database technology.

b. Design of Experiment

(1) Goal. The goal of the experiment was to gather data concerning the formulation of quiddity for data elements. The intent was to apply any new insights gained here to the design and execution of the next (primary) experiment.

(2) Experiment Packet. Two databases (overlapping in their real world domains), the Virus Database and the Hardware and Software Tracking System Database, designed by Naval Postgraduate students as class projects for a database management course, were used as the basis for the experiment. Twelve data elements from each database were selected for quiddity formulation. Care was taken to ensure that unique name violations did exist among the chosen data elements from each database. Each experiment packet contained the following: an overall information sheet, a work sheet (for practice and instructional purposes prior to beginning the experiment), a basic instruction sheet, a blank answer sheet, a general vocabulary list (words were not separated into quiddity component areas), a list of data dictionary entries pertaining to the selected data elements, and sample reports displaying the data captured by the selected data elements. A sample of this packet is contained in Appendix A.

(3) Procedure. Prior to beginning the experiment, a general overview of the thesis objectives was presented to the students. Each student was given an experiment packet, three students associated with each of the two databases. Next, the students were asked to read the general information sheet which included the purpose, background, details concerning quiddity concept and definitions, and examples. Then, the students were provided with instruction on the concept as well as on the representation and rules for quiddity formulation. Additionally, a work sheet of sample quiddity problems was provided and discussed with the students. The students were allowed to ask questions in order to clarify the concept.

Detailed instructions were provided to the students on the conduct of the actual experiment. Each student was asked to formulate quiddities for the twelve data elements provided using any and all information provided in the packet. They were asked not to discuss their answers with the other students nor to seek assistance from them. The students were not required to construct the quiddity expressions using the representation outlined in Section A of this chapter. We were more interested in the terms themselves. To avoid confusion, students were required to annotate quiddity terms using a table format. They were asked to provide comments pertaining to their "thought process" when developing the quiddities. Additionally, they were asked to comment on any areas of the concept which seemed difficult or confusing. It was suggested that they use only the vocabulary provided in the vocabulary list. If the vocabulary list did not contain a word which the student felt was crucial to forming the correct quiddity, they were instructed to add this word to the vocabulary and support its selection with a written justification. There was no set time limit for completion. Students were allowed to take the experiment packets with them and return them upon completion. This experiment was loosely controlled in order to gather as much raw input as possible.

c. *Results*

The goal of this experiment was to investigate several aspects of quiddity acquisition and formulation which led to the following questions. First, were the quiddities developed by the students correct? Second, did the students understand the concept and apply it correctly? Third, were the quiddities developed by students working with the same database identical?

The experiment results[12] were divided into two groups. The quiddities pertaining to the Virus Database were placed in Group 1 and the quiddities pertaining to the Hardware and Software Tracking System (HSTS) Database were placed in Group 2. There are a total of 36 quiddities in each Group, three for each of the twelve data elements. The correct quiddity[13] of each data element was compared with the quiddities developed by the students. TABLE I shows summary statistics of the correctness of the quiddities in each Group.

[12] All experiment results are contained in Appendix A.

[13] A master list of "correct" quiddities was developed prior to the experiment.

TABLE I QUIDDITY CORRECTNESS -- PRELIMINARY EXPERIMENT

*(TOTAL POSSIBLE MATCHES = 36)	Group 1	Group 2
Correct *Quiddity* Matches	0/36 (0%)	7/36 (19%)
Correct *Stuff* Matches	7/36 (19%)	24/36 (67%)
Correct *Stuff Attribute* Matches	24/36 (67%)	14/36 (39%)
Stuff Attribute Matching Correct *Stuff*	5/36 (14%)	6/36 (17%)
Stuff Matching Correct *Stuff Attribute*	0/36 (0%)	5/36 (14%)

The results suggest that the students did not understand the concept so were not able to apply it correctly. Few quiddities were correctly defined, i.e., there were no matches[14] between the correct quiddity and the experiment quiddities in Group 1, and only seven matches (out of a possible 36) in Group 2. Comparisons by quiddity component also showed some interesting trends. For the most part, the students were not able to correctly identify the *stuff* nor were they able to identify the *stuff attribute*. In fact, there were some instances where the students confused the *stuff* with the *stuff attribute*. Figure 7 shows specific instances of this confusion taken from the results.

[14] In order to be counted as an *exact match*, the experiment quiddities must be identical, term for term, to the "correct" quiddity.

Examples of data element *stuff* and *stuff attribute* confusion:

	BOOT-SECTOR (Group 1)	**VENDER** (Group 2)
Master	indicator(damage)	name(vender)
	vs.	vs.
Experiment	damage(virus)	vender(name)

NOTE: Notation = stuff attribute(stuff)

Figure 7 Stuff and Stuff Attribute Confusion

The quiddity comparisons within each Group reflected the same difficulties previously noted. There were no exact matches[15] between the three quiddities for each data element in either Group. Likewise, the definitions of the *stuff* and *stuff attribute* components were seldom in agreement. TABLE II shows statistics of the sameness of the quiddities in each Group.

TABLE II QUIDDITY SAMENESS -- PRELIMINARY EXPERIMENT

*(TOTAL POSSIBLE MATCHES = 12)	Group 1	Group 2
Exact *Quiddity* Matches	0/12 (0%)	0/12 (0%)
Exact *Stuff* Matches	4/12 (33%)	6/12 (50%)
Exact *Stuff Attribute* Matches	2/12 (17%)	2/12 (17%)

Student comments taken from discussions and written notes in the experiment packets also indicate confusion in applying the concept. The method for determining the *stuff* and

[15] In order to be counted as an *exact match*, all three quiddities for the data element must be identical. Likewise, when counting *exact matches* between quiddity components, all three components for that data element must be identical.

stuff attribute were not structured enough. The distinction between the *stuff* and *stuff attribute* component was unclear (see example in Section A.3.a of this chapter). This led to the inversion of both terms. Another problem area centered around the level of detail of the terms. For example, should the *stuff* term be **vehicle** or **truck** (assuming both words are included in the vocabulary)? If **vehicle** is the correct *stuff* term, the term **truck** could be the *stuff type*. Conversely, if **truck** is the correct *stuff* term, the term **vehicle** is unnecessary, i.e., the term provides no additional meaning. *Arity* also caused a great deal of confusion. Most students seemed at a loss when it came to determining the *arity* of a *stuff* term.

2. Refined Concept

We experienced difficulties similar in nature to those indicated by the initial experiment results when developing the master quiddities for the experiment. Clearly, the quiddity acquisition process requires refinemənt. The chief problem areas center around the lack of clear distinction between the *stuff* and *stuff attribute* components. This uncertainty led to confusion in discerning the *arity* of the *stuff* component and in identifying the sortal information provided by the *stuff type* and the *stuff attribute type*. Additionally, the level of detail required (e.g., virus vs. software) is unclear.

How can these problems be resolved? Clearly, a more descriptive definition of the quiddity components is needed. In other words, what is the meaning of each component and what kinds of information are each meant to supply? We propose that this concept can be clarified by examining quiddity from a linguistic perspective.

a. *A Linguistic Perspective*

Linguistics[16] is the science of language. Linguists divide knowledge about language into four overlapping components: the lexicon,[17] phonology,[18] syntax,[19] and semantics.[20] We are interested in the grammatical context of syntax and semantics as they apply to quiddity. The following discussion will draw a parallel between the structure of sentences and quiddity.

(1) Sentence Structure. A sentence consists of a linear sequence of words, one following the other. This composition of words follows regular patterns, otherwise known as syntactic rules, or grammar. Word order is important in English because it is an "analytic language," which means that the relationships of words in a sentence are indicated by the order in which the appear (Barnett 1964, 29).

The two essential parts of every sentence are an *actor* (subject) and an *action* (verb). Without these two parts, the meaning or semantics would be unclear. The normal order of these parts in a simple English sentence is subject/verb. The subject is what a sentence is about. The verb expresses what action the subject does. To find the verb in a sentence we often first find the subject. To find the subject, we ask the questions "*Who* or *what* is the sentence about?" or "*Who* or *what* is doing something in the sentence?" Then, we name the subject and ask

[16] From the *American Heritage Dictionary*, linguistics is "The study of the nature and structure of human speech."

[17] From the *American Heritage Dictionary*, lexicon is "The morphemes of a language." A morpheme is "A meaningful linguistic unit consisting of a word, such as *man*, or a word element, such as *-ed* of *walked*, that cannot be divided into smaller meaningful parts."

[18] From the *American Heritage Dictionary*, phonology is "The science of speech sounds, ..."

[19] From the *American Heritage Dictionary*, syntax is (*Gram.*) "The way in which words are put together to form phrases and sentences."

[20] From the *American Heritage Dictionary*, semantics is "The study or science of meaning in language forms, ..."

the questions "Did what?" or "Does what?" to find the verb. All other words in the sentence radiate from this subject/verb core. (Osborn 1989, 15-20)

The subject/verb core and all other words in a sentence can also be defined by the eight parts of speech (Osborn 1989, 67). They are:

1. **Noun:** any of a class of words naming or denoting a person, place, or thing, idea, quality, etc.
2. **Verb:** any of a class of words expressing action, existence, or occurrence; any phrase or construction used as a verb.
3. **Pronoun:** a word used in the place of or as a substitute for a noun.
4. **Adjective:** any of a class of words used to limit or qualify a noun or *substantive* (a word or group of words "subbing" as a noun).
5. **Adverb:** any of a class of words used to modify the meaning of a verb, adjective, or other adverb, in regard to time, place, manner, means, cause, degree, etc.
6. **Preposition:** a relational word that connects a noun, pronoun, or noun phrase to another element of the sentence, such as a verb, a noun, or an adjective.
7. **Conjunction:** a word used to connect words, phrases, clauses, or sentences.
8. **Interjection:** Wow! Phew! A word expressing emotion or simple exclamation, thrown into a sentence without grammatical connection.

These definitions will also be used in describing the structure of quiddity.

(2) Quiddity Structure. Similar to a sentence, quiddity consists of a linear sequence of components (rather than words), one following the other. Within components, the terms (if there are more than one) are listed alphabetically (a syntactic rule). Unlike sentences, where the relationships of words are indicated by their order, the relationship of quiddity components are indicated by their definitions. Because we are dealing with a *formal language*, component order, once designated, will never change. Component order is therefore irrelevant to determining meaning. For example, consider the quiddity *cost(truck)*. If the *formal language* has designated the component order to be *stuff attribute(stuff)*, then we know with certainty that *cost*

is the *stuff attribute* and *truck* is the *stuff*. What does this mean? We must know the relationship between the components (their definitions) to grasp the meaning of the sequence of components. If we know that *stuff* tells us what the data element is about and that the *stuff attribute* tells us what it is <u>about the *stuff*</u> we are interested in, then we can glean the quiddity's meaning. With this relationship defined, we now know that the data element captures information about *the cost of a truck*. Conversely, consider the quiddity *truck(cost)*. Given that the *formal language* has designated the component order to be *stuff(stuff attribute)* and the definitions for *stuff* and *stuff attribute* stated earlier still apply, we will derive the same meaning. The data element with this quiddity also captures information about *the cost of a truck*. It is important to note that component order is irrelevant in providing meaning only so long as we know for a certainty which component is which.

As in a sentence, quiddity has two essential components, *stuff* and *stuff attribute*, without which, there would be no meaning. All other quiddity components qualify the *stuff* and *stuff attribute*, just as all other words in a sentence qualify the subject and verb. Additionally, a parallel can be seen between the "questions" associated with the *stuff/stuff attribute* and subject/verb. (Figure 8)

<u>Description</u>

Stuff/Stuff Attribute:	***Subject/Verb:***
* ***Stuff --***	* ***Subject --***
"What is it about?"	"*Who* or *what* is the sentence about?"
****Stuff Attribute --***	* ***Verb --***
"What is it <u>about the *stuff*</u> are you interested in?"	"(the subject) Did what?" "(the subject) Does what?"

Figure 8 Parallel Between Stuff/Stuff Attribute and Subject/Verb

b. *Linguistics Applied to Quiddity Acquisition*

It is our premise that certain aspects of sentence structure, when applied to quiddity, will yield a more descriptive definition of the *stuff* and *stuff attribute* components. The key to developing the "correct" quiddity lies in correctly identifying the *stuff* and *stuff attribute* components. These components form the core of meaning for quiddity, just as the subject and verb form the core of a sentence. As presently defined, the *stuff* component is comprised of only one value or term while the *stuff attribute* is comprised of one or more values or terms. This ambiguity can be reduced by restricting both components to one and only one value or word phrase per quiddity definition. This parallels sentence structure in that a simple sentence has one and only one subject and verb.

Even though we have reduced the scope of values for both components, we are still faced with the lack of a clear distinction between them. The quiddity acquisition process requires that one first determine the value of the *stuff* component. Once the *stuff* is determined, the value of the *stuff attribute* can be captured. While developing the master list of quiddities for the initial experiment, we found that often the first value to become apparent was actually the *stuff attribute*. This led to confusion because the first inclination was to apply the value to the *stuff* component. This, of course, will result in an incorrect quiddity definition.[17] How can we modify the method to allow for first determining the *stuff attribute*? The first step is to further define the *stuff attribute*. If we compare all the *stuff attribute* values for each of the data elements in the master quiddity list of the initial experiment, we find that the values have a common characteristic. Each *stuff attribute* is a type of **MEASURE** of the *stuff* component. For example,

[17] An example illustrating the reversal of the *stuff* and *stuff attribute* component was provided in Section A.3.a. of this chapter.

in the quiddities name(software), cost(truck), and tail_number(aircraft), each *stuff attribute*, (i.e., name, cost, and tail_number), is a measure of the *stuff*, (i.e., software, truck, and aircraft, respectively).

To find this **MEASURE**, we first view the actual data contained in the field of the data element. Then we classify the data by grouping the collection under a general heading or name which answers the question "What is it?" or "What are these?" The aim is to categorize the actual words, codes, numbers, etc., that we see in the field. We are not concerned with what the data are representative of in the physical or concrete sense. We are looking for an abstract noun not a concrete noun.[22] The data in the field is a **MEASURE** of **SOMETHING**. The **MEASURE** is the *stuff attribute* and the **SOMETHING** is the *stuff*. Consider the following examples. Suppose a list of the data corresponding to values in a data element field appears as in Figure 9, Example (a).

Example (a):	Example (b):
[$12.95]	[sofa]
[$16.50]	[chair]
[$18.75]	[TV]
[$26.75]	[table]
[$33.56]	[desk]

Figure 9 Examples of Data Element Values

What values do we actually see in the field? We see a list of dollar amounts. What word can we use to categorize these amounts? Do they have a common characteristic? We can group them together and classify the amounts as ***prices*** or ***costs***. Therefore, (choosing one of the terms) the *stuff attribute* is *cost* (assuming this term is included in the vocabulary list). Now that we have

[22] A *concrete noun* is the name of anything physical, anything that can be touched, seen, heard, smelled, or otherwise perceived by the senses and occupies space. An *abstract noun* is the name of a quality, state, or action. It is an idea, and so may not be touched, seen, heard, smelled, or otherwise perceived by the senses. (Osborn 1989, 19)

found the *stuff attribute* we can determine the *stuff*. The data in the field is the *cost* of **something**. The **something** is the *stuff*. It could be the *cost* of **trucks**, **cars**, **ships**, etc.

Now consider the list of data shown in Example (b) in Figure 9. What do we see in this field? We see a list of furniture. What word can we use to categorize this list? We might be tempted to say that the category of this data is "furniture" but we would be wrong. Our aim is to capture a **measure** of the data, not what the data represents in the physical sense. What are these lists of "words" we see in the field? They are ***names***. Therefore, the *stuff attribute* is *name*. We now have *names* of **something**. *Names* of what? **Furniture**. So, the *stuff attribute* is *name* and the *stuff* is *furniture*. Of course, we will also have the data dictionary, a vocabulary, and any other available information to aid in defining these components.

We can now apply the "questions" in the original method to these examples as a verification of our answers. The two procedures will complement each other. If the values chosen for the *stuff* and *stuff attribute* components comply with both methods, the chances of incorrectly defining either component will be minimal. Figure 10 steps through the original method supporting our selection of values in the examples above.

Original Method

Description: This variable captures information about the cost of a truck.

What is it about? **A truck** *(stuff)*.

What is it about the **truck** are we interested in? Its **cost**.

Description: This variable identifies a specific piece of furniture.

What is about? **Furniture** *(stuff)*.

What is it about the **furniture** are we interested in? The **name** *(stuff attribute)*.

Figure 10 Quiddity Acquisition -- Samples of Original Method

Additionally, the remaining quiddity components, *stuff type*, *arity argument(s)*, and *stuff attribute type*, as well as *stuff* and *stuff attribute*, can be likened to various parts of speech.[23] Both the *stuff* and *stuff attribute* values are usually indicated with nouns. However, a subtle difference is that the *stuff attribute* will generally be represented by an abstract noun while the *stuff* is represented by a concrete noun. *Stuff types*, which qualify *stuff*, and *stuff attribute types*, which qualify *stuff attributes*, are indicated by adjectives or adverbs. Both *stuff types* and *stuff attributes* may have more than one value in a quiddity definition. This occurs when there are more than one qualifying terms for the *stuff* or *stuff attribute* as shown in the example depicting the quiddity of an unmanned fighter aircraft.[24] Both *unmanned* and *fighter* further describe the *stuff* term *aircraft* and are adjectives. There may be instances where a term is needed to further describe a *stuff type* or *stuff attribute type* term. If this occurs, the term, generally an adverb, is annotated as an additional *stuff type* or *stuff attribute type* term (as appropriate). When a *stuff* term has *arity*, its *argument(s)* will typically be represented by a noun(s).

We have suggested several changes in the quiddity acquisition process based on a linguistic approach. The refined concept for quiddity acquisition presented above is summarized below.

1. Gather Information. Examine the definition of the data element using the data dictionary and any other available information.

2. Examine Data. Examine a collection or list of actual data values contained in the data element field.

3. Classify Data. Classify the data by grouping the collection under a general heading or name which answers the question "What is it" or "What are these?" Each piece of data is an instance of the same thing or quality of something. The data is a type of **MEASURE** of **something**. This **MEASURE** is the *stuff attribute*.

[23] See Chapter III, Section B.2.a.(1) for a list of the eight parts of speech.

[24] See Chapter III, Section A.2.b.(1).

4. Find *Stuff Attribute* and *Stuff*. The *stuff attribute* measures **something**. The ***something*** is the *stuff*. The *stuff* will generally be a noun which is the object of a prepositional phrase associated with the *stuff attribute*. For example, if *cost* is the *stuff attribute*, answering the question "*Cost* of what?" will lead to the *stuff* term. The **what** is the *stuff* term. The **what** is also the object of the prepositic. -' phrase of what associated with the *stuff attribute* term *cost*.

5. Verify terms. Verify the *stuff* and *stuff attribute* terms by referring to the "questions" in the original method. What is the data element about? The *stuff*. What is it about the *stuff* we are interested in? The *stuff attribute*. If the terms also satisfy these questions, continue defining the remaining quiddity components as described in the original method. If not, return to the first step and begin again.

6. Define Remaining Components. Answer the questions "What sort of *stuff* is it? The *stuff type*. "What sort of *stuff attribute* is it?" The *stuff attribute type*. Is the *stuff* term a function of something else? If yes, determine the *argument(s)*.

The refined concept presented in this section addressed the problem of a lack of clear distinction between the *stuff* and *stuff attribute* components in the original concept. We have yet to address the remaining problem areas identified in the preliminary experiment, namely, the confusion in discerning the *arity* of *stuff*, the sortal information provided by the *stuff type* and *stuff attribute type*, and the level of detail required when defining components. These issues will be discussed in Chapter IV based on data obtained in the primary experiment.

C. QUIDDITY MANIPULATION AND INFERENCING

Our aim in this section is to present rules for determining quiddity equivalence. Based on these rules, we present several quiddity comparison procedures, both of which are necessary in examining the feasibility of quiddity in support of automatic detection of possible naming problems.

1. Rules for Quiddity Equivalence

Recall Bhargava et al.'s (1990) premise, that "*if two syntactically distinct variables have the same or equivalent dimension and quiddity, a **possible** unique names violation is indicated.* (Recall also that when checking for quiddity equivalence between data elements, we will assume that their corresponding dimensions are equivalent.) The rules for determining quiddity equivalence (presented below) are based on the following hypotheses:

H1. *Stuff* and *stuff attribute* are the most crucial quiddity components.

H2. Some use more specific terms than others when defining quiddities, e. g., vehicle : truck.

H3. People developing quiddities are likely to confuse the values defining *stuff type* with the values defining *arity*[25] (*arguments*).

H4. Some define quiddities more extensively than others, e. g., two values for *stuff type* vs. one value for *stuff type*.

a. *Term Equivalence*

What constitutes quiddity equivalence? An obvious answer is that quiddities are equivalent when they are syntactically identical, term for term. In other words, quiddities are equivalent when all quiddity components are equivalent. When are quiddity components equivalent? Again, an obvious answer is that the components are equivalent when the terms or values in the components are equivalent. We now reach the core of the equivalence process. When are terms equivalent? Obviously, terms are equivalent when they are syntactically identical, e.g., *cost* is equivalent to *cost*. However, suppose the terms being compared for equivalence are *price* and *cost*. Are these terms equivalent? The words are synonyms and, as such, their meanings are equivalent. Another aspect of equivalence appears when we compare the terms *vehicle* and *truck*. Are these terms equivalent? A *truck* <u>is a</u> *vehicle*. One term is simply more specific than the other. *Vehicle* could refer to a *truck*, but it could also refer to a *bus*. If we say that these terms are equivalent when they are really different (e.g., vehicle means bus), we run the risk of identifying a possible naming problem when it does not exist (Type I error). However, if we say the terms are not equivalent when they really are (e.g., vehicle means truck) and <u>do not</u> identify a possible naming problem, we run the risk of <u>not</u> identifying a problem when there really is one (Type II error). Since we are attempting to detect ***possible*** naming problems, we need to minimize

[25] Here, and in subsequent references, we are using the word *arity* to denote the *arguments* which are often needed to further describe *stuff*.

all errors, but specifically, Type II errors. We do not want to miss detecting a possible naming problem. Our premise is that in order to minimize Type II errors, we need the following three basic rules for term equivalence:

1. Two terms are equivalent if they are identical, i.e., match syntactically.
2. Two terms are equivalent if they are synonyms.
3. Two terms are equivalent if one term is a specialization of the other in the sense that all objects in the class represented by that term are also present in the class of objects, e. g., truck : vehicle (from H2).

We can now use these three term equivalence rules, singly or in combination, when determining component equivalence. We have divided the quiddity components into two sets for comparison. One set, designated the **Stuff Set**, contains the components *stuff*, *arity*, and *stuff type*. The other set, designated the **Stuff Attribute Set**, contains the components *stuff attribute* and *stuff attribute type*. Our premise is that quiddities are equivalent if and only if their corresponding **Stuff Sets** and **Stuff Attribute Sets** are equivalent (from H1). The following sections present equivalence rules for each set.

b. Stuff Set Equivalence

Stuff Set equivalence is defined in terms of equivalence of its components. The most evident and restrictive equivalence rule, alternative 1, is to require all components within the set to be equivalent (based on the term equivalence rules above) in order to have Stuff Set equivalence. Based upon the problem areas noted in the preliminary experiment, namely, the confusion in discerning the *arity* of *stuff*, the sortal information provided by the *stuff type*, and the level of detail required when defining components, this rule is too restrictive and would most likely result in a high number of Type II errors (H2 and H4 apply here). For example, the problem areas noted above will cause quiddities to be developed inconsistently. Even though the Stuff Sets of two data elements should be equal (because the data elements actually represent the same information), the inconsistency in defining the components would cause this rule to fail. Our premise is that we can alter this rule in order to compensate for these inconsistencies (in lieu of further refining the quiddity acquisition process). Consider the following example. The data element *purchase_cost* in Database A with the quiddity *purchase(cost(Dodge,used(truck(month))))* is compared to the data element *truck_cost* in Database B with the quiddity *purchase(cost(used(truck(month))))*. The quiddity of the data element *truck_cost* is certainly less specific than that of *purchase_cost*. Does this mean that the data elements do not actually represent the same information? We can not be sure without further examination, so we would want these data elements flagged as a *possible* naming violation. For that to happen, their quiddities (and thus their Stuff Sets) must be determined to be equivalent.

Our rule still states that Stuff Sets are equivalent if their three components are equivalent. However, now the rules for *arity* and *stuff type* equivalence must be altered to the following. (The *stuff* components are determined to be equivalent based upon the term equivalence rules.) Alternative 2 is that the *arity* of two data elements is equivalent if the *arity arguments* of one data element are contained in the *arity arguments* of the other data element. Similarly, the *stuff type* components are equivalent if the *stuff type* terms of one data element are contained in the *stuff type* terms of the other data element. For example, the *stuff type* term ***used*** (belonging to the data element *truck_cost*) is contained in the set of *stuff type* terms ***Dodge*** and ***used*** (belonging to the data element *purchase-cost*). Therefore, the *stuff type* components of the data elements are equivalent. It should be noted that term_1 <u>is contained in</u> a set of terms if term_1 is equivalent to a term in the set based upon the term equivalence rules listed above. (Additionally, an empty set is contained in any set.)

We can be even less restrictive in determining equivalence by combining the *arity arguments* and *stuff type* terms of the Stuff Set into one set and comparing this set for equivalence. Alternative 3 is that the Stuff Sets are equivalent if the *stuff* components of the Stuff Sets are equivalent (based on the term equivalence rules above) and the set of *arity* *arguments* and *stuff type* terms of one Stuff Set is contained in the set of *arity arguments* and *stuff type* terms of the other Stuff Set. Taking this one step further by combining the terms of all of the Stuff Set components into one set, alternative 4 is that the Stuff Sets are equivalent if the terms in the combined set of one Stuff Set are contained in the set of combined terms of the other Stuff Set.

c. *Stuff Attribute Set Equivalence*

The rationale presented above also applies to Stuff Attribute Set equivalence. Again, the most evident equivalence rule, alternative 1, is to require all components within the set to be equivalent (based on the term equivalence rules above) in order to have Stuff Attribute Set equivalence. Our hypothesis, that confusion between components and differing levels of description detail in quiddity acquisition can be compensated for by altering the equivalence rules, applies here as well. Alternative 2 is that the Stuff Attribute Set is equivalent if the *stuff attributes* are equivalent (based on the term equivalence rules defined earlier) and the *stuff attribute types* are equivalent. *Stuff attribute type* components are equivalent if the *stuff attribute type* terms of one data element are contained in the *stuff attribute type* terms of the other data element. Further, by combining the terms of all of the Stuff Attribute Set components into one set, alternative 3 is that the Stuff Attribute Sets are equivalent if the terms in the combined set of one Stuff Attribute Set are contained in the set of combined terms of the other Stuff Attribute Set.

2. Quiddity Comparison Procedures

In the preceding section, we defined several sets of quiddity equivalence rules. Various comparison procedures can be defined by applying these rules in different combinations. For clarity in discussion, these rules are depicted in Figure 11 in an abbreviated notation and are divided into sets of numbered rules.

EQUIVALENCE RULES:

Set A - Term Equivalence

($\equiv \longrightarrow$ "is equivalent to")

$\text{Term}_1 \equiv \text{Term}_2$ if rule 1, 2, or 3 is true.

1. Syntactic:
   $\text{Term}_1 \equiv \text{Term}_2$ if they match syntactically.

2. Synonym:
   $\text{Term}_1 \equiv \text{Term}_2$ if rule 1 is true OR if
   Term_1 and Term_2 are synonyms.

3. Network:
   $\text{Term}_1 \equiv \text{Term}_2$ if rule 1 is true OR if
   rule 2 is true OR if
   Term_1 and Term_2 are within the same
   classification network.

Set B - Stuff Set Equivalence

$\text{Stuff Set}_1 \equiv \text{Stuff Set}_2$ if rule 1, 2, 3, or 4 is true.

1. Plain and Simple:
   $\text{Stuff Set}_1 \equiv \text{Stuff Set}_2$ if

$$\begin{aligned} \text{stuff}_1 &\equiv \text{stuff}_2 \\ \text{arity}_1 &\equiv \text{arity}_2 \\ \text{stuff\_type}_1 &\equiv \text{stuff\_type}_2 \end{aligned}$$

2. Partial is-contained-in:
   (a $\rightarrow/\leftarrow$ b $\longrightarrow$ "a is contained in b or b is contained in a")

   $\text{Stuff Set}_1 \equiv \text{Stuff Set}_2$ if

$$\begin{aligned} \text{stuff}_1 &\equiv \text{stuff}_2 \text{ and} \\ \text{arity}_1 &\rightarrow/\leftarrow \text{arity}_2 \text{ and} \\ \text{stuff\_type}_1 &\rightarrow/\leftarrow \text{stuff\_type}_2 \end{aligned}$$

3. Part/Full is-contained-in:
   $\text{Stuff Set}_1 \equiv \text{Stuff Set}_2$ if

$$\begin{aligned} \text{stuff}_1 &\equiv \text{stuff}_2 \text{ and} \\ (\text{arity} + \text{stuff\_type})_1 &\rightarrow/\leftarrow (\text{arity} + \text{stuff\_type})_2 \end{aligned}$$

4. Full is-contained-in:
   $\text{Stuff Set}_1 \equiv \text{Stuff Set}_2$ if

   $(\text{stuff} + \text{arity} + \text{stuff\_type})_1 \rightarrow/\leftarrow (\text{stuff} + \text{arity} + \text{stuff\_type})_2$

Set C - Stuff Attribute Set

$\text{Stuff Attribute Set}_1 \equiv \text{Stuff Attribute Set}_2$
if rule 1, 2, or 3, is true.

1. Plain and Simple:

$$\begin{aligned} \text{stuff\_attribute}_1 &\equiv \text{stuff\_attribute}_2 \text{ and} \\ \text{stuff\_attribute\_type}_1 &\equiv \text{stuff\_attribute\_type}_2 \end{aligned}$$

2. Partial is-contained-in

$$\begin{aligned} \text{stuff\_attribute}_1 &\equiv \text{stuff\_attribute}_2 \text{ and} \\ \text{stuff\_attribute\_type}_1 &\rightarrow/\leftarrow \text{stuff\_attribute\_type}_2 \end{aligned}$$

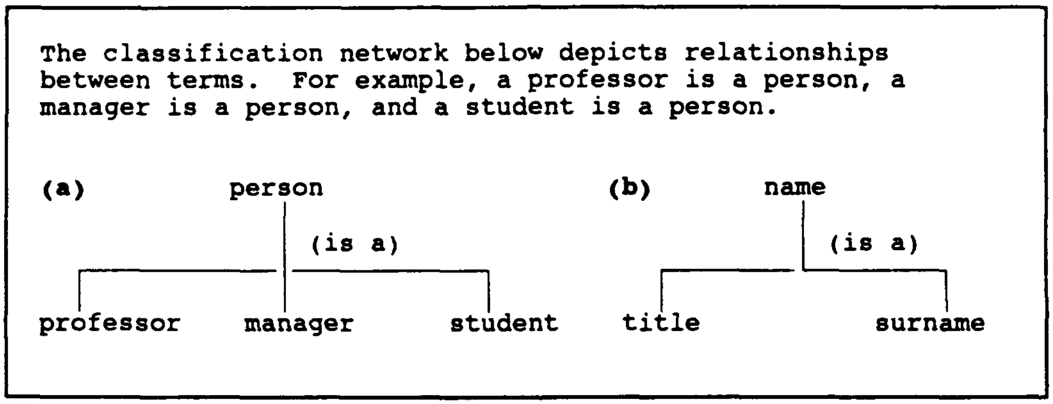
3. Full is-contained-in

$$(\text{stuff\_attribute} + \text{stuff\_attribute\_type})_1 \rightarrow/\leftarrow (\text{stuff\_attribute} + \text{stuff\_attribute\_type})_2$$

Figure 11 Quiddity Equivalence Rules

a. Term Equivalence Rule Set

We defined three basic term equivalence rules in the previous section. These basic rules are applied in three distinct combinations. Rule A1 (Set A, Rule 1) states that two terms are equivalent if they are syntactically identical, or in other words, they are a syntactic match. Rule A2 states that two terms are equivalent if they are synonyms <u>or</u> if they are a syntactic match. Rule A3 states that two terms are equivalent if they are related, as in a classification network, if they are synonyms, <u>or</u> if they are a syntactic match. Figure 12 illustrates the organization of terms into a classification network.



The classification network below depicts relationships between terms. For example, a professor is a person, a manager is a person, and a student is a person.

Figure 12 Classification Network

b. Stuff Set Equivalence Rule Set

We defined four Stuff Set equivalence rules as shown in Set B of Figure 11. We chose to maintain a strict equivalence rule in most combinations for the *stuff* component due to its significance in the quiddity definition (from H1). Set B component equivalency rules are based on the rules in Set A, e.g., components are equivalent if their terms are equivalent. Rule B1 states that Stuff Sets are equivalent if each of their components are equivalent. Rule B2 states that two Stuff Sets are equivalent if their *stuff* components are equivalent, the *arity arguments* of one are contained in the *arity arguments* of the other, and the *stuff type* terms of one are contained in the *stuff type* terms of the other.[26] Rule B3 states that two Stuff Sets are equivalent if their *stuff* components are equivalent and the combined set of *arity* and *stuff type* terms of one are contained in the combined set of *arity* and *stuff type* terms of the other. Rule B4 states that two Stuff Sets are equivalent if the combined set of *stuff*, *arity*, and *stuff type* terms of one are contained in the combined set of *stuff*, *arity*, and *stuff type* terms of the other. The stricture of these rules can be reduced slightly by varying the term equivalence rules.

c. *Stuff Attribute Equivalence Set*

We defined three Stuff Attribute Set equivalence rules as shown in Set C of Figure 11. We chose to maintain a strict equivalence rule in most combinations for the *stuff attribute* component due to its significance in the quiddity definition (from H1). Set C component equivalency rules are also based on the rules in Set A. Rule C1 states that Stuff Attribute Sets are equivalent if each of their components are equivalent. Rule C2 states that two Stuff Attribute Sets are equivalent if their *stuff attribute* components are equivalent and if the *stuff attribute type* terms of one are contained in the *stuff attribute type* terms of the other. Rule C3 states that two Stuff Attribute Sets are equivalent if the combined set of *stuff attribute* and *stuff attribute type* terms of one are contained in the combined set of *stuff attribute* and *stuff type* terms of the other. Again, the stricture of these rules can be reduced slightly by varying the term equivalence rules.

d. *Procedures*

The three sets of rules can be combined into twelve distinct procedures. This is best shown in a matrix format. (See TABLE III) For each procedure, there must be one term equivalence rule, one Stuff Set equivalence rule, and one Stuff Attribute Set equivalence rule. In the matrix, both Rule B2 and Rule B3 (Stuff Set) are combined with Rule C3 (Stuff Attribute

[26] It should be noted that term_1 <u>is contained in</u> a set of terms if term_1 is equivalent to a term in the set based upon the term equivalence rules. Additionally, an empty set is contained in any set.

Set) in the separate procedures because Rules B2 and B3 more closely match in equivalence concepts with Rule C2 than they do with Rule C3. The idea is maintain equivalence consistency between the Stuff Set and the Stuff Attribute Set. For example, it is not consistent to apply the most strict rule of equivalence to the Stuff Set (e.g., Rule B1) while at the same time applying the loosest equivalence rule to the Stuff Attribute Set (e.g., Rule C3) in determining quiddity equivalence.

TABLE III EQUIVALENCE PROCEDURES

Component Equivalence		Term Equivalence Rules		
Set B	Set C	A1	A2	A3
B1	C1	A1,B1,C1	A2,B1,C1	A3,B1,C1
B2	C2	A1,B1,C1	A2,B2,C2	A3,B2,C2
B3	C2	A1,B3,C2	A2,B3,C2	A3,B3,C2
B4	C3	A1,B4,C3	A2,B4,C3	A3,B4,C3

These twelve procedures were applied to, and tested using a prototype application developed in Prolog[27]. A given procedure is specified simply by specifying the appropriate rules within each set. These procedures are examined in greater detail in Chapter IV.

[27] The prototype program listing, along with the data listing, can be found in Appendix B.

IV. PRIMARY EXPERIMENT

A. DESIGN OF EXPERIMENT

1. Subjects

The same six Naval Postgraduate students who participated in the preliminary experiment also participated in the primary experiment. These students were selected in order to take advantage of their experience in quiddity formulation. The intent of this selection was to eliminate any "noise" in the experiment (due to not understanding the concept) which could interfere with the analysis of the concept itself.

2. Goal

The goal of this experiment was to gather data concerning the formulation of quiddity for data elements using the refined concept described in Chapter III, Section B.2. These quiddities would then be compared using the procedures developed in Chapter III, Section C, to determine if the concepts were equivalently applied by the subjects and if the quiddities could be useful in support of automatic detection of unique name violations.

3. Experiment Packet

Two databases (overlapping in their real world domains), the Naval Postgraduate School Automated Catalog (NAC) and the Course Requirements and Forecasting Tool (CRAFT), designed by Naval Postgraduate students as class projects for a database management course, were used as the basis for this experiment. Fifteen data elements from each database were selected for quiddity formulation. Care was taken to ensure that unique name violations did exist among the chosen data elements from each database. Each experiment packet contained the following: all information included in the preliminary experiment packet to include students' original responses with the addition of "suggested quiddity answers," a new information sheet , an updated work sheet with answers to examples provided in the preliminary experiment, a basic instruction sheet, a blank answer sheet, a vocabulary list (words were separated into quiddity component areas), a list of data dictionary entries pertaining to the selected data elements, and sample reports displaying the data captured by the selected data elements. An example of this packet is contained in Appendix C.

4. Procedure

The procedure for this experiment was similar to the preliminary experiment in most respects. Prior to beginning the experiment, a general overview of the thesis objectives was again presented to the students. Each student was given an experiment packet, and were associated with each of the two databases. Group integrity remained constant from the preliminary experiment when assigning students to a database. Next, the students were asked to read the new information sheet which included the purpose, a review of details concerning quiddity concept and definitions, and a new approach to be used in addition to the original concept. Then, the students were provided with instruction on the new approach as well as a review of the original concept. Additionally, an updated work sheet with answers to the sample quiddity problems used during preliminary experiment instruction was provided and discussed with the students. The students were allowed to ask questions in order to clarify the concept. Responses to the preliminary experiment were discussed and further instruction was provided on the concept of arity.

Conduct of the experiment was closely matched to that of the preliminary experiment[28] with the following exceptions. Each student was asked to formulate quiddities for fifteen data elements (as opposed to twelve data elements in the preliminary experiment) and to provide comments on the usefulness of the refined approach in quiddity formulation. Students were also asked to comment on any areas of the concept which remained difficult or confusing.

[28] See Chapter III, Section B.1.b.(4)

Unlike the preliminary experiment, the students were restricted to using only the vocabulary provided in the vocabulary list. If the vocabulary list did not contain a word which the students felt was crucial to forming the correct quiddity, they were instructed to provide comments to that effect but to complete all quiddities to the best of their ability using only the vocabulary in the list. The vocabulary was restricted in order to increase control over the experiment, thus more effectively testing the concept.

B. EXPERIMENT RESULTS

The goal of this experiment was to investigate several aspects of quiddity acquisition and formulation, with emphasis on the problems noted in the preliminary experiment, and to test the hypothesis noted in Chapter III, Section C. Specific areas of interest are highlighted by the following questions. Did the refined concept improve the distinction between the *stuff* and *stuff attribute* components, i.e., were their values still subject to inversion? Was the idea of *arity* understood and correctly applied? Did the equivalence procedures compensate for any of the problems noted in the preliminary experiment?

1. Quiddity Formulation

The experiment results[29] were divided into two groups. The quiddities pertaining to The Naval Postgraduate School Automated Catalog (NAC) Database were placed in Group 1 and the quiddities pertaining to the Course Requirements and Forecasting Tool (CRAFT) Database were placed in Group 2. There are a total of 45 quiddities in each Group, three for each of the fifteen data elements. The correct quiddity[30] of each data element was compared with the quiddities developed by the students. TABLE IV shows summary statistics of the correctness of the quiddities in each Group.

[29] All experiment results are contained in Appendix C.

[30] A master list of "correct" quiddities was developed prior to the experiment.

TABLE IV QUIDDITY CORRECTNESS -- PRIMARY EXPERIMENT

*(TOTAL POSSIBLE MATCHES = 45)	Group 1	Group 2
Correct *Quiddity* Matches	13/45 (29%)	15/45 (33%)
Correct *Stuff* Matches	39/45 (87%)	35/45 (78%)
Correct *Stuff Attribute* Matches	25/45 (56%)	27/45 (60%)
Stuff Attribute Matching Correct *Stuff*	0/45 (0%)	1/45 (2%)
Stuff Matching Correct *Stuff Attribute*	0/45 (0%)	4/45 (9%)

The results suggest that the students have a much better understanding of the concept. There was a significant increase in the number of correctly defined quiddities in both Groups, more than double the percentage correctly defined in the first experiment.[31] Comparisons by quiddity component (*stuff* and *stuff attribute*) also improved greatly. Based upon comments from the students, this overall improvement can be attributed to several factors. First, all students indicated that the refined concept simplified and added clarity to the quiddity acquisition process. Two students stated that they used only the refined concept in determining the quiddity definitions, i.e., they did not use the original concept to verify their definitions. Second, all students reported that the restrictive vocabulary reduced the uncertainty in defining the quiddities. Third, all students related that their familiarity with the concept eased the task of defining the quiddities in this experiment.

The quiddity comparisons within each Group improved overall. There were still very few exact matches between the three quiddities for each data element in either Group. For the

[31] In order to be counted as an *exact match*, the experiment quiddities for the data elements must be identical, term for term, to the "correct" quiddity.

most part, the reason the quiddities were not exact matches was due to differences within *stuff type* and *stuff attribute type*. This reflects uncertainty in the level of detail required in defining quiddity and supports our fourth hypothesis (see Chapter III, Section C). Some students demonstrated a tendency to be consistently more specific than others. The number of exact matches within the *stuff* and *stuff attribute* components increased significantly from the first experiment. These improvements can also be attributed to the same factors discussed in connection with quiddity correctness. TABLE V shows summary statistics of the sameness of the quiddities developed within each Group.[32]

TABLE V QUIDDITY SAMENESS WITHIN GROUPS -- PRIMARY EXPERIMENT

*(TOTAL POSSIBLE MATCHES = 15)	**Group 1**	**Group 2**
Exact *Quiddity* Matches	3/15 (20%)	0/15 (0%)
Exact *Stuff* Matches	11/15 (73%)	10/15 (67%)
Exact *Stuff Attribute* Matches	6/15 (40%)	6/15 (40%)

Arity continues to cause a great deal of confusion. Students remain at a loss when it comes to determining the *arity* of a *stuff* term. In both Groups, *arity* was correctly identified by only one student. It should be noted that the data pertaining to *arity* can be misleading. There are only three data elements in the experiment (one in the NAC database and two in the CRAFT database) which have an *arity* greater than 0 and require defining. Most students indicated that they left the *arity* component blank because they were not certain if the *stuff* component had *arity* greater than 0. This resulted in an *arity* "correctness" of 73% for Group 1 and 80% for Group 2 because 27 of the 30 data elements in the experiment have *arity* of 0!

[32] A match here means that all three subjects in the same group used the exact same term(s).

2. Procedures For Quiddity Comparison

The data in the experiment was compared for equivalence using the twelve procedures described in Chapter III, Section B. TABLE VI (taken from Chapter III, Section C) provides an overview of the rule combination for each procedure. The set designation is now indicated by the position of the rule number. For example, a procedure number now consists of just three numbers, i.e., 243. The number in the first position (2) indicates rule number 2 from Set A. The number in the second position (4) indicates rule number 4 from Set B. The last number (3) indicates rule number 3 in Set C.

TABLE VI EQUIVALENCE PROCEDURES

Component Equivalence		Term Equivalence Rules		
Set B	Set C	A1	A2	A3
B1	C1	111	211	311
B2	C2	122	222	322
B3	C2	132	232	332
B4	C3	143	243	343

The experiment data consists of six sets of quiddities, three from the NAC Database and three from the CRAFT Database. Our experiment assumes that we are planning to integrate the two databases. Our goal is to detect possible naming problems (synonyms and homonyms) by comparing the quiddities for each database using the above procedures. There are a total of nine unique pair-wise combinations of quiddities (each of the three sets of CRAFT quiddities compared with each of the three sets of NAC quiddities). For each combination, there are 225 comparisons of data elements (15 x 15). The quiddities within each database were also compared with each other in order to provide data pertaining to the "sameness" of the quiddities. Finally, the master quiddity list for each database were compared.

There were 192 database comparisons performed ((9 x 12) + (6 x 12) + 12) with a grand total of 42,525 comparisons (when counting each data element comparison). The prototype implementation produces a listing for each database comparison. The report lists pairs of data elements which may have naming violations. These pairs of data elements are categorized as possible homonyms or synonyms. Sample output listings are in Appendix C. These comparisons (225 for each procedure plus 12 for the master quiddity comparison) were subdivided by procedure and type (e.g., between databases, within database, and master) and analyzed to determine the number of Type I and Type II errors. The raw data is compiled in TABLES located in Appendix C. An analysis of this data is presented in the next section.

C. ANALYSIS OF COMPARISON PROCEDURES

The objective of this section is to determine the combination of equivalence rules which will minimize Type I and Type II errors (with priority on Type II errors). There are a total of five synonyms and three homonyms in this experiment.

1. Synonyms

The number of Type II errors decreased or remained constant as the term equivalence rules became lax.[33] The broader definition of equivalence increased the chances of correctly identifying all naming violations. As the component equivalence rules were broadened, the Type II errors decreased, but not at a very significant rate. (However, notice that there are only 5 true synonym problems.) Conversely, as the term equivalence rules became lax and the component equivalence rules broadened, Type I errors increased. Clearly, it is more important to prevent Type II errors, than it is to avoid increasing Type I errors. However, the results do indicate that the "middle of the road" procedures are best, i.e., those procedures using component

[33] A procedure is more "lax" than another procedure if the following rule is true.
Given Procedure_1 $(i_1 j_1, k_1)$, Procedure_2 $(i_2 j_2, k_2)$, and "more lax" ⟶ "< <" :

$\text{Proc}_1 << \text{Proc}_2$ if
$i_1 \leq i_2$ and
$j_1 \leq j_2$ and
$k_1 \leq k_2$ (where at least one is a strict inequality)

set rules 22 or 32. The "best" procedure is the one with the lowest error rate (both Type I and II errors). By providing weights to each type of error, we can choose the procedure with the lowest error rate.[34] Even though the experiment quiddities were, on the average, only 31% correct, the trends in numbers of Type I and II errors very closely paralleled those of the master list. This seems to indicate that there may not be just "one" correct quiddity for a data element. Additionally, there is no difference between component equivalence rules 22 and 32. This seems to indicate that either *arity* is irrelevant or that the results are skewed. (The fact that 27 of the 30 data elements have no *arity* could skew these results.) (See Figures 13, 14, 15, and 16)

2. Homonyms

Since this thesis focused primarily on the synonym problem, homonyms will only be addressed briefly. Homonyms appear to be a much simpler problem to detect than do synonyms because it is necessary to compare quiddities only when two data element names are syntactically identical. However, the same methods apply once the identical names are detected.

The number of Type II errors increased as the term equivalence rules became lax. The broader definition of equivalence increased the chances of failing to identify all .aming violations. As the component equivalence rules were broadened, the Type II errors increased significantly. Type I errors were nonexistent throughout. To identify a Type I error, the procedure would have to incorrectly determine that equivalent quiddities are not equivalent while at the same time detecting identical data element names. This circumstance appears to be a rare occurrence. All our experimental results point to the conclusion that the best procedure for detecting homonyms is the one which is the most strict, i.e., not lax. (See Figures 17 and 18)

[34] Given the total number of Type I errors, N_I, and the total number of Type II errors, N_{II}, and weights, W_I and W_{II}, respectively, then the error rate for the procedure is:

$$\text{error-rate}(N_I, N_{II}) = W_I \cdot N_I + W_{II} \cdot N_{II} \quad (W_I \text{ will normally be less than } W_{II})$$

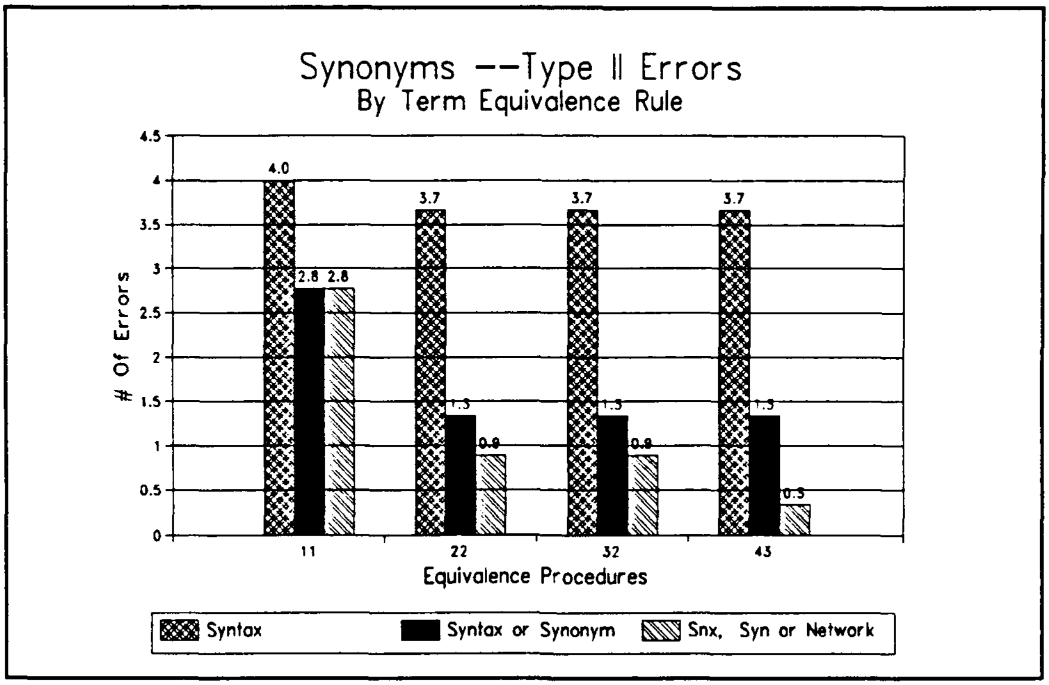


Figure 13 Synonyms -- Type II Errors (Experiment)

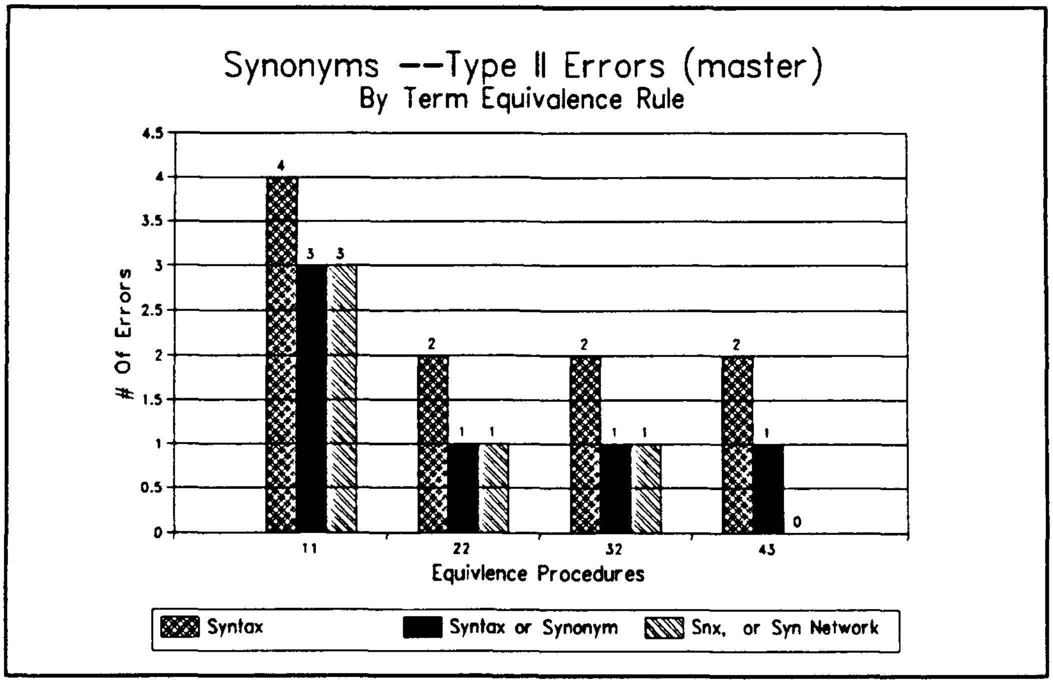


Figure 14 Synonyms -- Type II Errors (Master)

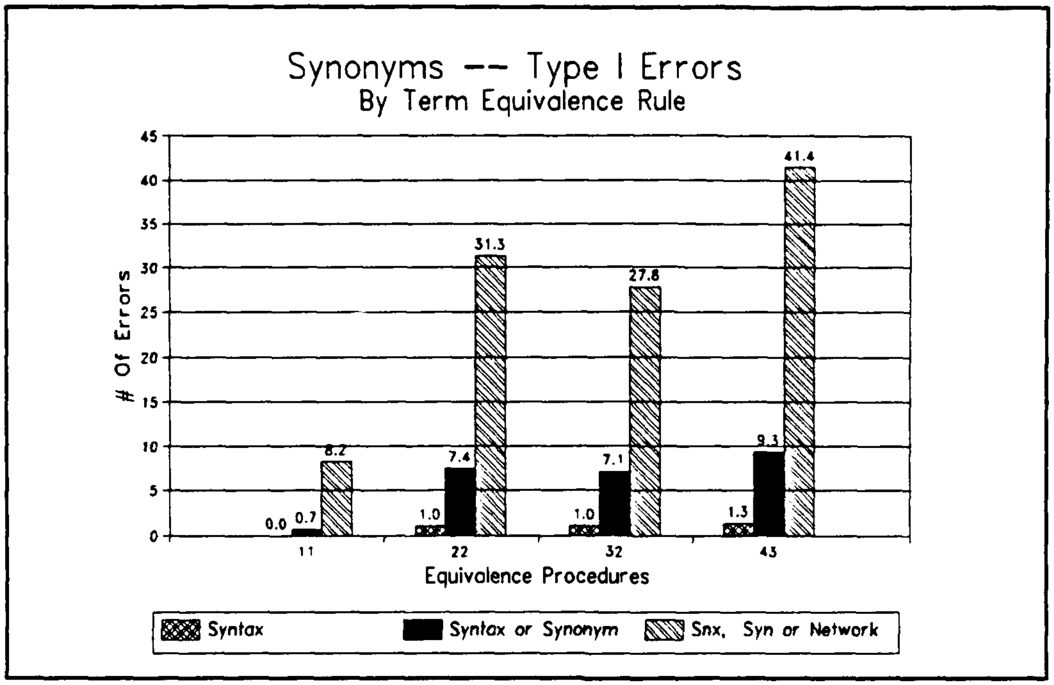


Figure 15 Synonyms -- Type I Errors (Experiment)

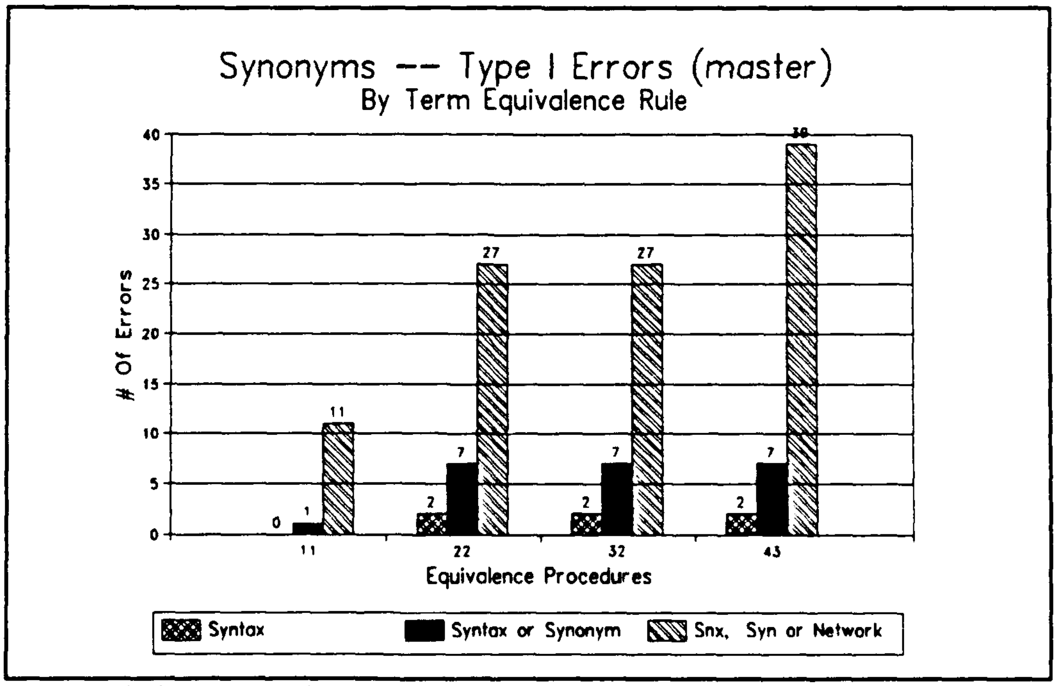


Figure 16 Synonyms -- Type I Errors (Master)

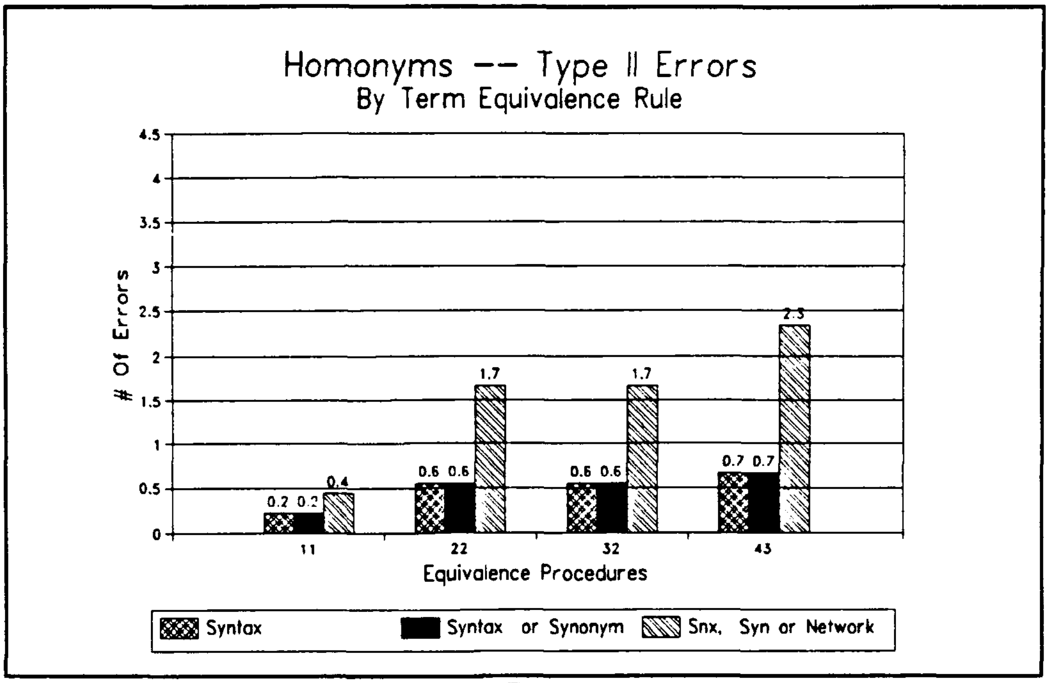


Figure 17 Homonyms -- Type II Errors (Experiment)

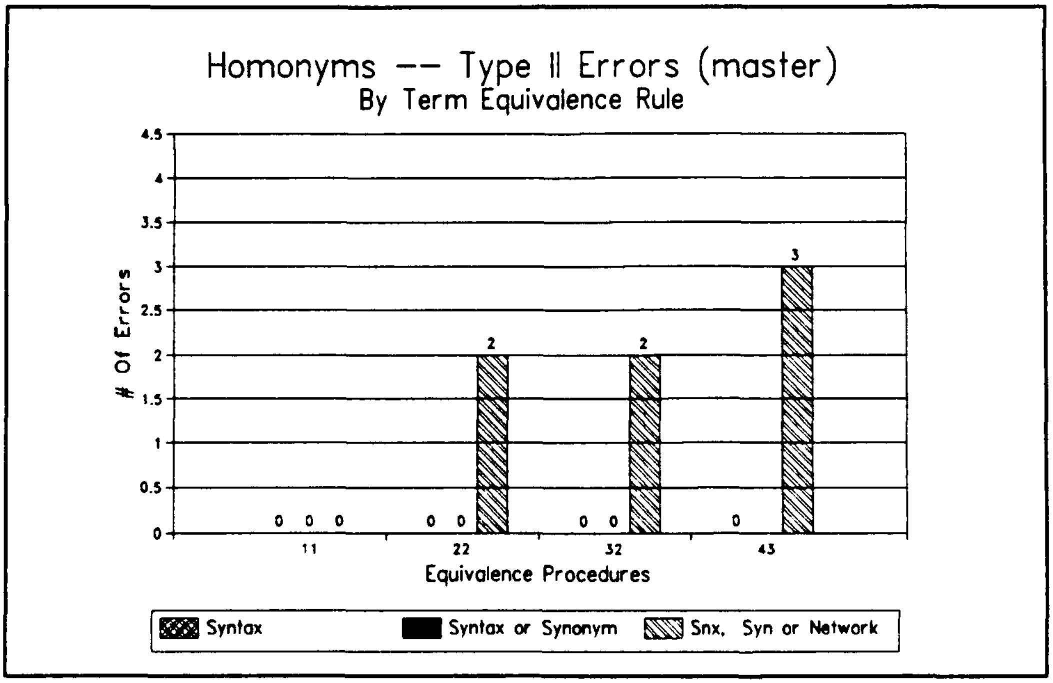


Figure 18 Homonyms -- Type II Errors (Master)

V. CONCLUSION

A. CONTRIBUTIONS AND LIMITATIONS

This thesis has examined and enhanced a method for automatically detecting possible naming problems of data elements prior to database integration. We explored several aspects of quiddity, namely, quiddity concept definition, quiddity acquisition, and quiddity manipulation and inferencing procedures. Specifically, we administered the first real, experimental application of the concept of quiddities. With a careful analysis and extensive examples, the concept was refined and adapted to the database environment. Our research also constitutes the first important study on quiddity acquisition. We investigated how the issues of vocabulary, synonyms, classification properties, and degrees of specificity affect quiddity acquisition. Finally, we developed, implemented, and tested a number of alternative inference procedures, along with equivalence rules, for use in automatically detecting possible naming problems.

Our research indicates that the concept of quiddity can be applied in the database context to provide a basis for automatically detecting unique name violations. Experiment results show that two individuals seldom consistently develop syntactically identical quiddities for the same data elements. However, we found that by varying the rules for equivalence, these differences in defining quiddities could be compensated for, ultimately resulting in equivalent quiddities (as they were initially supposed to be). The use of a specific vocabulary, coupled with the use of synonyms significantly countered this problem of inconsistency. Conversely, the use of classification properties tended to exacerbate this problem. However, the size and number of databases limits the scope of our conclusions. Additionally, our experiments were not fully controlled. This fact aided our efforts in gathering as much data as possible, but limits our ability to advance any firm, fully supported conclusions. Our research does provide an indication of the direction in which full, formal testing should follow.

We also presented several inference procedures for evaluating quiddity equivalence. Initial results indicate that the best procedure for detecting synonyms is one that lies (approximately half way) between those procedures with the most strict and the most lax equivalence rules. On the other hand, indications are that homonyms are best found utilizing a procedure with very strict equivalence rules.

The concept of quiddity is a complex issue. Correct and consistent application of this concept depends upon a clear and unambiguous understanding of each of the components comprising quiddity. Clearly identifying each component with more descriptive names would facilitate comprehension of the concept. For example, the word "quiddity" succinctly and appropriately describes the semantic information being captured. However, the words "stuff" and "stuff attribute" are vague, unclear descriptions of the quiddity components. Therefore, we propose the following name changes in future applications of this concept.

1. **Stuff.** *Stuff* describes what the data element is about. All other quiddity components revolve around this description as it is the heart of the quiddity definition. A more appropriate and descriptive title is "**gravamen**." From *Roget's II, The New Thesaurus*, gravamen is "The most central and material part."

2. **Stuff Attribute.** *Stuff attribute* is a <u>measure</u> of the *stuff* component. Based on the name suggested for the *stuff* component, a fitting and more specific title is "**gravamen measure**."

3. **Stuff Type.** *Stuff type* further describes *stuff*. Following the recommendations above, a more suitable title would be "**gravamen type**" or "**gravamen qualifier**."

4. **Stuff Attribute Type.** *Stuff attribute type* further describes *stuff attribute*. Similarly, a pertinent title is "**gravamen measure type**" or "**gravamen measure qualifier**."

B. ISSUES FOR FURTHER RESEARCH

There are several issues for consideration in further research. Formal testing of the processes in quiddity acquisition is needed as the model has yet to be validated. The quiddity inferencing procedures should be further developed and tested on a more extensive database. Additionally, the prototype can be refined and improved to increase efficiency. More in depth analysis of the linguistic aspects is feasible. Could it lead to a theory? Development of an interactive system to support quiddity declarations would aid in quiddity acquisition. For example, the system would check validity of the quiddity definitions and provide alternatives (e.g., if dimension = currency, then the stuff attribute = cost, price, value ...). Finally, can the concept of quiddity be helpful in identifying different representation conflicts, in addition to naming conflicts? In summary, the concept of quiddity, in addition to demonstrating usefulness in detecting naming problems in database integration, may also be useful in detecting or resolving other conflict areas in database integration.

APPENDIX A -- PRELIMINARY EXPERIMENT

This Appendix contains samples of the contents of the packet given to the students during the preliminary experiment. Additionally, a TABLE with each students' experiment results (quiddity definitions) along with the master quiddities for this experiment is included. Items specified above are found on the following pages:

EXPERIMENT #1

A. PURPOSE

The title of my proposed thesis is "The Problem of Unique Names Violations in Database Integration." The general area of research will be experimenting with a proposed method for automatically detecting possible naming problems of data elements prior to database integration. The purpose of this experiment is to gather data to assist me in analyzing this proposed method.

B. BACKGROUND

A database can be defined as "a store of integrated data capable of being directly addressed for multiple uses; ... ."[1] The data in a database are stored in units called data elements. Each data element has a unique name associated with it. For example, the data element which contains an individual's social security number could be called "SSN." Data elements also have assigned data types (i.e., integer, character, etc.) and field lengths.

As databases continue to grow and develop, the number of uses for the databases also increase. To support this growth, a need to integrate/combine databases has appeared. One aspect which must be dealt with before integration can occur is the problem of naming conflicts in like data elements. Specifically, the problem deals with two or more data elements having different names in each database but containing information about the same thing. For example, one database might call the data element which contains a social security number, "SSN," while another calls it "SSNO." Before these two databases can be merged, the naming conflict must be resolved.

How do we find these conflicts? Clearly, we need more semantic information: **information about what the data element represents.** There are two basic methods currently used in identifying these conflicts. The first method is a syntactic check: they check the data element names syntactically or match data types (i.e. integer, character, etc.) or field lengths. The second method involves a screen of the data dictionary. The data dictionary has more descriptive information about the data elements but is written in natural language, which is not useful for machine inference. The proposed method contained in my thesis involves further defining

[1] Elias M Awad, *Management Information Systems: Concepts, Structure, and Applications* (Menlo Park, California: The Benjamin Cummings Publishing Company, Inc., 1988), p. 593.

each data element by providing dimensional information and information about the nature or essence (quiddity) of the data contained in the data element. By comparing the dimensional information and quiddity of data elements in databases to be integrated, we hope to easily identify any naming conflicts which exist. The primary emphasis for the experiment concerns developing the "quiddity" of data elements.

C. QUIDDITY

"Quiddity" is the name given to the description of what information is captured by the data element. For example, you might have a data element named "cost." You can probably surmise that the data element contains the cost of something, but what is that something? If we knew the quiddity of this data element, we would know what the something is.

1. Components of Quiddity

In order to use a computer program to compare the quiddity of data elements, we need to have a standard way of recording it without writing it in natural language form. For example, let's suppose that the data element "cost" captures information about the "retail cost of an IBM personal computer."
We must dissect this definition into parts, almost like diagramming a sentence. When you diagram a sentence, you list the subject, adjectives, adverbs, and verb etc. When you determine quiddity, you must list the "stuff, stuff types, stuff attributes, stuff attribute types, and arity."

a. Stuff

"Stuff" answers the question "What is the data element about?," or put another way, it is the subject of the description. Stuff is usually indicated by a noun, describing individual things or collections of individual things, i.e., cars, trucks, ships, etc. In the above example, the stuff of the data element "cost" is "personal computer."

b. Stuff Type

"Stuff type" answers the question "What sort of or kind of stuff is it? Stuff types are usually indicated with an adjective but can also be indicated by a noun. Stuff types further describe stuff. For example, with both stuff and stuff type we can distinguish between a "truck tire" and a "tire truck." In the first case, what is the data element about? It is about a tire. What sort of tire? A truck tire. Thus the stuff is tire and the stuff type is truck. However, in the second case, the data element is about a truck. What sort of truck? A tire truck. Thus the stuff is truck and the stuff type is tire. In our example above, the stuff type of "personal computer" (stuff) is "IBM."

c. *Stuff Attributes*

"Stuff attributes" answer the question "What is it about "the stuff" that you are interested in? Stuff attributes are usually indicated with nouns. What is it about a "personal computer" that we are interested in? The cost. So "cost" is the "stuff attribute" of "personal computer" (stuff).

d. *Stuff Attribute Types*

"Stuff attribute types" answer the question "What sort of stuff attribute is it?" Stuff attribute types usually qualify measurements and are typically indicated with nouns. From above, the stuff attribute was "cost." What sort of "cost" are we interested in? Retail cost. Thus "retail" is the stuff attribute type of the stuff attribute "cost."

e. *Arity*

When a term has "arity," it can be defined by one or more arguments. "Arity" is a term more commonly used in mathematics in conjunction with functions. For example, the function of "addition" has an arity of "2" because you must have two arguments in order to perform the function, in other words, to add. Division also has an arity of 2 whereas the square root function has an arity of 1 (you only need one argument to find the square root). With quiddity, we use arguments, when necessary, to further define "stuff." For example, some stuff expressions may have no arguments, i.e., truck, ship, computer, etc., and would have an arity of "0." We do not need any further definitions to know what a ship or a computer is. However, suppose "path" is the stuff expression. In this case, we would need to know the two end points of the path in order to define the exact path. Thus, "path" has an arity of "2" since it has two arguments (the two end points). In our example with the data element "cost," the stuff expression has an arity of "0."

2. Notation

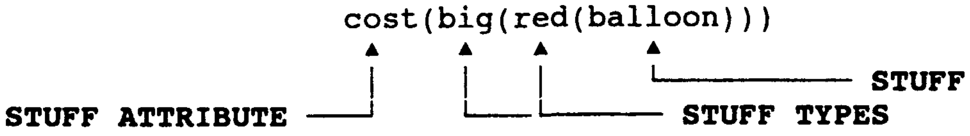
Now that we have defined all the components of quiddity, we must have a way of recording the information. In general, quiddity notation will take the following form:

Stuff Attribute Type(Stuff Attribute(Stuff Type(Stuff(Argument 1, Argument 2, ... Argument N))))

There may be instances where there is more than one term for each category. When this happens, the terms should be listed alphabetically.

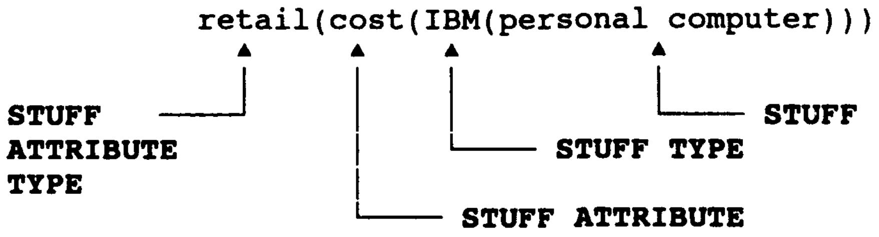
a. Example 1

Suppose you have a data element which captures information about the cost of a big red balloon. What is the data element about? A balloon (stuff). Does balloon need any arguments to define it? No, so "balloon" has an arity of "0" (no arity arguments). What sort of balloon (stuff) is it? It is big <u>and</u> red. We have two stuff types. What is it about the balloon that we are interested in? The cost (stuff attribute). What sort of cost is it? We don't know from the information given so we don't have a stuff attribute type. The quiddity for this example is:



b. Example 2

Let's look again at the data element "cost" which captures information about the "retail cost of an IBM personal computer." What is the data element about? A personal computer (stuff). Do we need any arguments to define personal computer? No. Thus there are no arity arguments (arity 0). What sort of personal computer (stuff) is it? It is an IBM (stuff type). What is it about the personal computer that we are interested in? The cost (stuff attribute). What sort of cost is it? Retail (stuff attribute type) cost. The quiddity is:



WORK SHEET[1]

1. How do we capture the "meaning" of what a data element represents?

a. A proposed method for capturing this "meaning" uses a type of formal "language" with various rules for forming the definition of the "meaning." This definition or description is called "quiddity."

> "From the *Oxford English Dictionary*, quiddity is 'The real nature or essence of a thing; that which makes a thing what it is.' Of course, ... [the proposed] language for expressing quiddities is only a model, or approximation, of genuine quiddity, if it exists."

Example 1:

● DATABASE 1

- Variable: *purchase_cost*
- Description:
  "Purchase cost of a truck"

● DATABASE 2

- Variable: *cost_of_purchase*
- Description:
  "Cost of purchase of a truck"

b. Let's begin defining the basic component of quiddity.

Example 1a:

● DATABASE 1

- Variable: *purchase_cost*
- Description:
  "Purchase cost of a truck"
- Dimension: currency
- Stuff: truck

● DATABASE 2

- Variable: *cost_of_purchase*
- Description:
  "Cost of purchase of a truck"
- Dimension: currency
- Stuff: truck

[1] All examples and quotes in this work sheet have been borrowed from the following reference: Bhargava, Hemant K., Steven O. Kimbrough, and Ramayya Krishnan, *Unique Names Violations: A Problem For Model Integration or You Say Tomato, I Say Tomahto* (University of Pennsylvania, Department of Decision Sciences and Carnegie Mellon University, SUPA, Working Paper, 1990), pp. 5-8.

2. Now, let's change the variables slightly.

a. Notice that the description changed but not the "dimension" or "stuff."

Example 2:

● DATABASE 1

- Variable: *purchase_cost*
- Description:
  "Cost of purchasing a truck"
- Dimension: currency
- Stuff: truck

● DATABASE 2

- Variable: *production_cost*
- Description:
  "Cost of producing a truck"
- Dimension: currency
- Stuff: truck

b. What is the *quiddity*?

Sample line of reasoning used in Example 2a to describe "quiddity."

> "Both variables are about the same stuff: trucks. They differ in what it is they represent *about* trucks. What is it about trucks they describe? Purchasing in one case and production in the other. What is it about purchasing and production that they represent? Cost, in both cases. And what about cost? Nothing else." This line of reasoning suggests the quiddity descriptions in Example 2a.

Example 2a:

● DATABASE 1
- Variable: *purchase_cost*
- Description:
  "Cost of purchasing a truck"
- Dimension: currency
- Quiddity: *cost(purchase(truck))*

  STUFF ATTRIBUTES ↑ ↑ ↑ **STUFF**
- Quiddity Paraphrase:
  "the cost of purchase of a truck"

● DATABASE 2
- Variable: *production_cost*
- Description:
  "Cost of producing a truck"
- Dimension: currency

  cost(production(truck))

  STUFF ATTRIBUTES ↑ ↑ ↑ **STUFF**
- Quiddity Paraphrase:
  "the cost of production of a truck"

3. Below is a list of several data elements in a "Home Inventory" database. See if you can define the quiddity for each data element.

DATA DICTIONARY EXCERPT:

FIELD NAME	TYPE	DESCRIPTION
ITEM	1{character}40	Identifies a specific piece of property, i.e., sofa, dining room chair, TV, etc.
QUANTITY	1{integer}3	Identifies the total number of like items or pieces of property owned, i.e., "2" if two sofas are owned.
VALUE	1{integer}8	Identifies the current replacement cost of a specific piece of property.
DATE	1{date}8	The month, day, and year the property was purchased or acquired.
PRICE	1{integer}8	Identifies the amount paid for a specific piece of property.
WEIGHT	1{integer}5	The total number of pounds a specific piece of property weighs.
FREE_WEIGHT	1{logical}1	Whether weight of a specific piece of property applies toward the professional weight allowance or not, i.e., "Y" if yes or "N" if no.

DATA ELEMENTS	QUIDDITY				
	STUFF	ARITY (ARGUMENTS)	STUFF TYPE	STUFF ATTRIBUTE	STUFF ATTRIBUTE TYPE
ITEM					
QUANTITY					
VALUE					
DATE					
PRICE					
WEIGHT					
FREE_WEIGHT					

EXPERIMENT #1

Instructions:

1. Determine the quiddity for each data element listed. Record the components of the quiddity in the appropriate columns of the row listing the data element. Please write legibly.

2. Please keep track of the order in which you determine the quiddity components for each data element by placing a number in the upper left corner of the appropriate "box" in the table. For example, if the first term you define for the first data element is its stuff, the second term is its stuff type, and the third term is its stuff attribute, the table would look like this:

DATA ELEMENT	STUFF	ARITY	STUFF TYPE	STUFF ATTRIBUTE	STUFF ATTRIBUTE TYPE
Computer	1 PC		3 IBM	2 COST	4 RETAIL

3. Each quiddity may or may not have a term for each component. (HINT: You will always have at least a "stuff" component and a "stuff attribute" component.) Some quiddities may have more than one term for a component. If there is more than one term, write both terms in the "box" and place its ordering number to the left of each term.

4. I am interested in the "method" you use in determining the quiddity, particularly in the "thought process" you go through in working through this experiment. Any comments or suggestions you have (even in bullet form) is appreciated.

COMMENTS:

VIRUS DATABASE

DATA ELEMENTS	QUIDDITY				
	STUFF	ARITY (ARGUMENTS)	STUFF TYPE	STUFF ATTRIBUTE	STUFF ATTRIBUTE TYPE
ALIAS					
BOOT_SECTOR					
COMMAND_COM					
DISINFECTANT					
EXE_FILES					
MACHINE_TYPE					
OPERATING_SYSTEM					
REFERENCE					
SIZE					
VENDOR					
VIRUS					
IBM					

HARDWARE AND SOFTWARE TRACKING SYSTEM DATABASE

DATA ELEMENTS	QUIDDITY				
	STUFF	ARITY (ARGUMENTS)	STUFF TYPE	STUFF ATTRIBUTE	STUFF ATTRIBUTE TYPE
COMPATIBLE					
DESCRIPT					
HLAN					
MAKE					
MODEL					
NAME					
OFFICE					
PUBLISHER					
REMARKS					
SITE					
SSERIAL					
VENDER					

VOCABULARY

addition

alias

brand

building

bytes

commercial

compatibility

compatible

component

computer

damage

destroy

disinfectant

disk

disk boot sector

executable files

file

general

hardware

IBM

identification

indicator

information

internal

LAN

literature

location

machine

manufacturer

model

name

network

number

office

operating system

piece

Publisher

receipt

reference

sector

serial

site

size

software

source

supplier

system

vender

vendor

virus

VIRUS DATABASE DATA DICTIONARY

FIELD NAME	TYPE	DESCRIPTION
ALIAS	1{character}20[1]	Commonly used alias
BOOT_SECTOR	1{character}1[2]	Whether or not the virus corrupts the disk boot sector
COMMAND_COM	1{character}1[2]	Whether or not the virus infects the system
DISINFECTANT	1{character}10[3]	Name of a commercially available virus disinfection routine which is known to successfully remove this virus
EXE_FILES	1{character}1[2]	Whether or not the virus infects EXE files
MACHINE_TYPE	1{character}10[3]	Name of a commercial computer type
OPERATING_SYSTEM	1{character}10[3]	Name of the operating system used
REFERENCE	1{character}80[1]	Significant literature reference for virus
SIZE	1{integer}5[4]	Size of virus in number of bytes
VENDOR	1{character}80[3]	Commercial source of disinfectant product
VIRUS	1{character}20[5]	Name of each virus which infects a computer
IBM	1{character}1[2]	Whether or not the computer system is IBM or IBM compatible

[1] Nulls allowed

[2] "y" or "n" only, no nulls

[3] No nulls

[4] Small integer, 0-32767

[5] Unique key, no nulls

HARDWARE AND SOFTWARE TRACKING SYSTEM DATABASE
DATA DICTIONARY

FIELD NAME	TYPE	DESCRIPTION
COMPATIBLE	1{character}1	Identifies a piece of software as being compatible with IBM or Apple
DESCRIPT	1{character}30	Identification type of a piece of hardware, i.e., keyboard, monitor, etc.
HLAN	1{logical}1	Identifies a piece of hardware as local area network compatible (True) or not (False)
MAKE	1{character}30	Identifies the brand of a piece of hardware
MODEL	1{character}15	Identifies the model number/type of a piece of hardware, i.e., "VGA" for a monitor or "286" for Zenith PC, etc.
NAME	1{character}30	Identifies the name of a piece of software
OFFICE	1{character}4	Identification number of an office that is inside a building
PUBLISHER	1{character}30	Identifies the name of a software publisher
REMARKS	1{character}80	General remarks about a piece of hardware
SITE	1{logical}1	Identifies a piece of software as having a site license (True) or not (False)
SSERIAL	1{character}25	Identifies the serial number of a piece of software
VENDER	1{character}30	Identifies the name of a hardware vender

Virus Database Listing
Updated 04 April 1990

Virus Name	Disinfector	Infection Area							Features			Damage Cause					
		P	H	F	O	E	C	I	M	E	Bytes	B	O	P	D	F	L
XA1	cleanup	n	n	n	n	n	y	n	n	y	1539	n	y	y	n	y	y
1392	cleanup	n	n	n	n	y	y	y	y	n	1392	n	y	y	n	n	y
1210	cleanup	n	n	n	n	n	y	n	y	n	1210	n	y	y	n	n	y
1720	cleanup	n	n	n	y	y	y	n	y	n	1720	n	y	y	n	y	y
Saturday 14th	cleanup	n	n	n	y	y	y	n	y	n	685	n	y	y	n	y	y
Korea	m-disk	n	y	y	n	n	n	n	n	n		y	y	n	n	n	n
Vcomm	cleanup	n	n	n	n	y	n	n	n	n	1074	n	y	y	n	n	y
ItaVir	cleanup	n	n	n	n	y	n	n	n	n	3880	y	y	y	n	n	y
Solano	cleanup	n	n	n	n	n	y	n	y	n	2000	n	y	y	n	n	y
V2000	cleanup	n	n	n	y	y	y	y	y	n	2000	n	y	y	n	n	y
1554	scan	n	n	n	n	y	y	y	y	n	1554	n	y	y	n	n	y
512	scan	n	n	n	n	n	y	y	y	n	0	n	y	y	n	n	y
EDV	m-disk	y	y	y	n	n	n	n	y	n		y	y	n	n	n	n
Joker	cleanup	n	n	n	n	n	y	y	y	n		n	y	y	n	n	n
Icelandic-3	cleanup	n	n	n	n	y	n	n	y	n	853	n	y	y	n	n	n
Virus-101	cleanup	n	n	y	y	y	y	y	y	y	2560	n	n	y	n	n	n
1260	cleanup	n	n	n	n	n	y	n	n	y	1260	n	n	y	n	n	n
Perfume	cleanup	n	n	n	n	n	y	n	n	n	765	n	n	y	n	n	n
Taiwan	cleanup	n	n	n	n	n	y	n	n	n	708	n	n	y	n	n	n
Chaos	m-disk	n	y	y	n	n	n	n	y	n		y	y	n	y	y	n
Virus-90	cleanup	n	n	n	n	n	y	n	y	n	857	n	n	y	n	n	n
Oropax	cleanup	n	n	n	n	n	y	n	y	n	2773	n	y	y	n	n	n
4096	cleanup	n	n	n	y	y	y	y	y	n	4096	n	y	y	y	n	y
Devil's Dance	cleanup	n	n	n	n	n	y	n	y	n	941	n	y	y	y	n	y
Amstrad	cleanup	n	n	n	n	n	y	n	n	n	847	n	n	y	n	n	n
Payday	cleanup	n	n	n	y	y	y	n	y	n	1808	n	n	y	n	n	n
Datacrime II-B	cleanup	n	n	n	n	y	y	y	n	y	1917	n	n	y	n	y	n
Sylvia	cleanup	n	n	n	n	n	y	n	n	n	1332	n	n	y	n	n	n
Do-nothing	cleanup	n	n	n	n	n	y	n	y	n	608	n	n	y	n	n	n
Sunday	cleanup	n	n	n	y	y	y	n	y	n	1636	n	y	y	n	n	n
Lisbon	cleanup	n	n	n	n	n	y	n	n	n	648	n	n	y	n	n	n
Typo	cleanup	n	n	n	n	n	y	n	y	n	867	n	y	y	n	n	n

Key -

INFECTION AREAS

P - disk partition table

H - fixed disk boot sector
F - floppy disk boot sector

O - .OVR files
E - .EXE files

C - .COM files
I - COMMAND.COM

FEATURES

M - remains memory resident
E - self encrypting

Bytes - virus size

DAMAGE CAUSED

B - corrupts disk boot sector
O - degrades system operation

P - corrupts .COM, .EXE, .OVR files
D - corrupts data files

F - formats part or all of disk
L - corrupts file linkage

Report of Software by Name and Version

/ersion	Publisher	License	Serial Number	Procurement Number	Date Received	Lan Compatible	Hardware Compatible
* Name AMI PRO							
2.00	SAMNA		766SD75S	89RQ23433	08/20/89	No	IBM
* Name C COMPILER							
5.00	MICRO SOFT	4590-34	879239342	89RQ34I	03/05/89	No	IBM
* Name DBASE III							
1.10	ASHTON TATE	23940044-4	99SAD	90RQ123K	02/05/90	No	IBM
* Name DBASE IV							
1.00	ASHTON TATE	9823-332-112	1001-02	89RQ1234	01/03/90	Yes	IBM
* Name DESKTOP PUBLISHER							
1.00	DIGITAL RESEARCH	9837548	185494	90RQ0330	04/01/90	No	IBM
* Name GEM DRAW							
2.00	DIGITAL RESEARCH	77-343	987244-211	90RQ234	01/02/90	No	IBM
* Name HARVARD GRAPHICS							
2.00	ALUS		398-24	87RQ334	03/05/87	No	IBM
* Name LOTUS 123							
2.00	LOTUS DEVELOPMENT CORP	7358-67-8863	4568-23	87RQ123E	03/10/87	No	IBM
* Name PFS: PROFESSIONAL WRITE							
3.00	PFS	83896	230096	89RQ1238	07/02/89	No	IBM
* Name PRESENTATION							
1.20	ALDUS		877-23	89RQ433R	04/19/89	No	IBM
* Name RENEX TMS							
3 [illegible]	RENEX	221922840	98-12339	87RQ8732	04/02/87	No	IBM
* N[illegible] RIGHT WRITER							
1.20	PFS		345-A349	88RQ34KD	02/13/88	No	IBM
* Name TIME-LINE							
4.00	SYMANTIC	13003-234-2333	2340-123-11111	90RQ12E2	01/02/90	Yes	IBM
* Name WINDOWS							
2.00	MICROSOFT	2134J218D9	77648766	90RQ0023	07/10/89	No	IBM

Report of Software that is LAN Compatible

Software Name	Publisher	Version
** Hardware Type IBM		
DBASE IV	ASHTON TATE	1.00
TIME-LINE	SYMANTIC	4.00

```
                              Report of Hardware Procurement by Procurement Number


Tag   Serial                                                                    Procurement Date
Number Number          Make            Model           Description       Internal Date        Received


** Procurement Number 87RQE1203

** Vendor ZENITH
00100  932WF0381TS2    ZENITH          286             KEYBOARD          No       03/12/87    03/12/87

** Procurement Number 89RQ3432

** Vendor COMPUADD
38     1051867         VENTEL          2400B           MODEM             No       05/06/89    05/06/89

** Procurement Number 89RQ345K

** Vendor HEWLET PACKARD
35     2841A1979       HEWLET PACKARD  AT COMP         CPU               No       05/09/89    05/09/89

** Procurement Number 89RQ980

** Vendor HEWLET PACKARD
36     61577553        HEWLET PACKARD  VGA             MONITOR           No       04/18/89    04/18/89

** Procurement Number 89RQE1234

** Vendor APPLE
4      F851EEXM5825    MACINTOSH       IIE             CPU               No       02/12/89    02/12/89

** Procurement Number 89RQE234

** Vendor APPLE
2      669944          MACINTOSH       IIE             KEYBOARD          No       02/15/89    02/15/89

** Procurement Number 90RQE3401

** Vendor ZENITH
2      933NE0306TOQ    ZENITH          286             MONITOR           No       01/12/90    01/12/90
```

VIRUS DATABASE
Subject A

DATA ELEMENTS	QUIDDITY				
	STUFF	ARITY (ARGUMENTS)	STUFF TYPE	STUFF ATTRIBUTE	STUFF ATTRIBUTE TYPE
ALIAS	virus		computer	name	alias
BOOT_SECTOR	disk-boot-sector			indicator	damage
COMMAND_COM	system			indicator	damage
DISINFECTANT	software		disenfectant commercial	name	
EXE_FILES	executable_files			indicator	damage
MACHINE_TYPE	computer		commercial	name	
OPERATING_SYSTEM	system		operating	name	
REFERENCE	literature		virus	reference	
SIZE	virus		computer	size	bytes
VENDOR	vendor		commercial disinfectant	name	
VIRUS	virus		computer	name	
IBM	system		computer	indicator	compatibility IBM

VIRUS DATABASE
Subject B

DATA ELEMENTS	QUIDDITY				
	STUFF	ARITY ARGUMENTS	STUFF TYPE	STUFF ATTRIBUTE	STUFF ATTRIBUTE TYPE
ALIAS	virus		information	alias	
BOOT_SECTOR	virus		information	damage	disk_boot_sector
COMMAND_COM	virus		information	damage	system
DISINFECTANT	virus		information	disinfectant	destroy
EXE_FILES	virus		information	damage	executable_files
MACHINE_TYPE	computer		information	brand	
OPERATING_SYSTEM	computer		information	operating_system	
REFERENCE	virus		information	reference	
SIZE	virus		information	size	
VENDOR	virus		information	disinfectant	Source
VIRUS	virus		information	name	
IBM	computer		information	operating_system	compatible IBM

VIRUS DATABASE
Subject C

DATA ELEMENTS	QUIDDITY				
	STUFF	ARITY (ARGUMENTS)	STUFF TYPE	STUFF ATTRIBUTE	STUFF ATTRIBUTE TYPE
ALIAS	virus		computer	name	alias
BOOT_SECTOR	category		damage	indicator	disk_ boot_sector
COMMAND_COM	category		damage	indicator	command_com
DISINFECTANT	disinfectant		virus	name	
EXE_FILES	category		damage	indicator	executable_ file
MACHINE_TYPE	computer			name	brand
OPERATING_SYSTEM	operating_ system			name	
REFERENCE	reference			information	identification
SIZE	virus		computer	size	bytes
VENDOR	vendor		disinfectant	name	
VIRUS	virus			indicator	
IBM	category		IBM computer		

HARDWARE AND SOFTWARE TRACKING SYSTEM DATABASE
Subject 1

DATA ELEMENTS	QUIDDITY				
	STUFF	ARITY (ARGUMENTS)	STUFF TYPE	STUFF ATTRIBUTE	STUFF ATTRIBUTE TYPE
COMPATIBLE	software		piece	compatible	IBM (Apple)
DESCRIPT	hardware		component	identification	
HLAN	hardware		component	compatible	lan
MAKE	hardware		component	brand	
MODEL	hardware		component	model	
NAME	software		piece	name	
OFFICE	number	building		office	
PUBLISHER	name			publisher	software
REMARKS	hardware		component	remarks	general
SITE	software	company location	piece	license	site
SSERIAL	software		piece	number	serial
VENDER	name	component	company	vendor	hardware

HARDWARE AND SOFTWARE TRACKING SYSTEM DATABASE
Subject 2

DATA ELEMENTS	QUIDDITY				
	STUFF	ARITY ARGUMENTS	STUFF TYPE	STUFF ATTRIBUTE	STUFF ATTRIBUTE TYPE
COMPATIBLE	software		piece	compatibility	vendor
DESCRIPT	hardware		component	identification	
HLAN	compatibility		lan hardware	indicator	
MAKE	hardware		component	brand	vendor
MODEL	hardware		component	model	number
NAME	software			name	
OFFICE	office		building	identification number	
PUBLISHER	name			vendor	software
REMARKS	hardware			information	general
SITE	license		site software	indicator	
SSERIAL	software		piece	serial number	
VENDER	name		company	vendor	hardware

HARDWARE AND SOFTWARE TRACKING SYSTEM DATABASE
Subject 3

DATA ELEMENTS	QUIDDITY				
	STUFF	ARITY (ARGUMENTS)	STUFF TYPE	STUFF ATTRIBUTE	STUFF ATTRIBUTE TYPE
COMPATIBLE	category		compatibility	indicator	
DESCRIPT	hardware		piece	infomation	general
HLAN	category		lan compatibility	indicator	
MAKE	hardware		piece	brand	
MODEL	hardware		piece	number identification	model
NAME	software			name	
OFFICE	office	building		number	
PUBLISHER	publisher		software	name	
REMARKS	hardware		piece	information	general
SITE	category		site license	indicator	
SSERIAL	software		piece	number	serial
VENDER	vender		hardware	name	

VIRUS DATABASE
MASTER

DATA ELEMENTS	QUIDDITY				
	STUFF	ARITY (ARGUMENTS)	STUFF TYPE	STUFF ATTRIBUTE	STUFF ATTRIBUTE TYPE
ALIAS	software		virus	name	alias
BOOT_SECTOR	damage	virus disk_boot_sector		indicator	
COMMAND_COM	damage	virus system		indicator	
DISINFECTANT	virus		disinfectant	name	
EXE_FILES	damage	virus executable_files		indicator	
MACHINE_TYPE	hardware			name	brand
OPERATING_SYSTEM	software		operating_system	name	
REFERENCE	reference		literature virus	name	
SIZE	virus			size	
VENDOR	vendor		software disinfectant	name	
VIRUS	software		virus	name	
IBM	compatibility	brand	hardware	indicator	

HARDWARE AND SOFTWARE TRACKING SYSTEM DATABASE
MASTER

DATA ELEMENTS	QUIDDITY				
	STUFF	ARITY ARGUMENTS	STUFF TYPE	STUFF ATTRIBUTE	STUFF ATTRIBUTE TYPE
COMPATIBLE	compatibility	brand	software	indicator	
DESCRIPT	hardware		component	name	
HLAN	compatibility		LAN	indicator	
MAKE	hardware			name	brand
MODEL	hardware		component	** *model_number*	
NAME	software			name	
OFFICE	office		internal	number	
PUBLISHER	publisher		software	name	
REMARKS	hardware			information	general
SITE	license		site software	indicator	
SSERIAL	software			** *serial_number*	
VENDER	vender		hardware	name	

** The words "model", "serial", and "number" were provided separately in the vocabulary list of experiment #1. Based on the comments received after experiment #1, the words should have been provided as a "word group" as shown in the table above. This combination better describes the stuff attribute.

APPENDIX B -- PROTOTYPE IMPLEMENTATION

This Appendix contains a copy of the Prolog program listing (the prototype, a list of the data used by the prototype (i.e., experiment and master data). Additionally, a sample report/list produced by the program is included. Items specified above are found on the following pages:

```
/* To run the system, here's what happens:
  1. Across DB Test:
     Perform step 2 for every pair of subjects
     (S1,S2) where S1 wrote quiddities for DB1
     and S2 wrote quiddities for DB2.
  2. For a given pair of subjects (S1,S2):
     perform the quiddity test with each pair
     of data elements (E1,E2) where E1 is in
     DB1 and E2 is in DB2.
     */

/* Problems with current implementation (3-5-91):


*/

/* Results:
    1. Identical element pairs. (Names and quiddities equal.)
    2. Synonym pairs.
    3. Homonym pairs.
    4. No action pairs. (Quiddities and names unequal.)
 */

/* Example:
     acrossDBtest(ProcNo,[(naf,[a,b,c]),(craft,[d,e,f])]).  */

     go :-
        write('Procedure No? '),read(ProcNo),
   nl,
   write('First Database Name? '),read(DB1),
   nl,
   write('First Subject Name? '),read(Subject1),
   nl,
   write('Second Database Name? '),read(DB2),
   nl,
   write('Second Subject Name? '),read(Subject2),
        acrossDBtest2(ProcNo,(DB1,Subject1),(DB2,Subject2)).



     acrossDBtest2(ProcNo,(DB1,Subject1),(DB2,Subject2)) :-
          quiddity(DB1,Subject1,Element11,_,_,_,_,_),
          quiddity(DB2,Subject2,Element21,_,_,_,_,_),
     quiddity_eq(ProcNo,[DB1,Subject1,Element11],
                             [DB2,Subject2,Element21],Result),
     fail.

     acrossDBtest2(ProcNo,(DB1,Subject1),(DB2,Subject2)) :-
         printreport(ProcNo,(DB1,Subject1),(DB2,Subject2)),
    retractall(tomato(_,_,_,_)).


/* For assertion of results of quiddity test. */


     determine(ProcNo,[DB1,Sub1,E1],[DB2,Sub2,E2],Answer,Assertion) :-
          (E1 = E2,
     Answer = yes,
     asserta(tomato(ProcNo,[DB1,Sub1,E1],[DB2,Sub2,E2],match));

     E1 = E2,
     Answer = no,
     asserta(tomato(ProcNo,[DB1,Sub1,E1],[DB2,Sub2,E2],homonym));
     not(E1 = E2),
     Answer = yes,
     asserta(tomato(ProcNo,[DB1,Sub1,E1],[DB2,Sub2,E2],synonym));

     not(E1 = E2),
     Answer = no,
     asserta(tomato(ProcNo,[DB1,Sub1,E1],[DB2,Sub2,E2],relax))).

 /*
RULES FOR EQUALITY:

*/

/* QUIDDITY EQUIVALENCE: equivalence of quiddities of two data elements. */

/* quiddity_eq(Element1,Element2,Answer).
Answer will be yes, or no.    */

quiddity_eq(ProcNo,E1,E2,Answer) :-
    (stuff_set_eq(ProcNo,E1,E2,yes),
    stuff_attribute_set_eq(ProcNo,E1,E2,yes),Answer = yes;
         Answer = no),
    determine(ProcNo,E1,E2,Answer,Assertion),!.


/* STUFF-SET EQUIVALENCE: equivalence of stuff-sets. */
/* stuff_set_eq(ProcNo,E1,E2,Answer). */
/* stuff-set(A) =~ stuff-set(B)
  if stuff_set_eq(ProcNo,A,B,yes) . */

/*   Set 2/Rule 1 -- The stuff-sets are equal if the stuff terms
        are equal, the arity terms are equal, and the
        stuff_type terms are equal.
        */

         stuff_set_eq((TE,1,SAE),A, B, yes) :-
             ProcNo = (TE,1,SAE),
        stuff(A,SA),stuff(B,SB),
        arity(A,ArA),arity(B,ArB),
        stuff_type(A,StA),stuff_type(B,StB),

        term_eq(ProcNo,SA,SB,yes),
             term_eq(ProcNo,ArA,ArB,yes),
             term_eq(ProcNo,StA,StB,yes).

/*   Set 2/Rule 2 -- The stuff-sets are equal if the stuff terms
        are equal, and the arity terms of one is
        contained in the arity  term of the other,
        and if the stuff type term of one is contained
        in the stuff type term of the other.
        */

         stuff_set_eq((TE,2,SAE),A, B, yes) :-
             ProcNo = (TE,2,SAE),
             stuff(A,SA),stuff(B,SB),
        arity(A,ArA),arity(B,ArB),
        stuff_type(A,StA),stuff_type(B,StB),

        term_eq(ProcNo,SA,SB,yes),
             contained_in_check(ProcNo,ArA,ArB,yes),
             contained_in_check(ProcNo,StA,StB,yes).

/*   Set 2/Rule 3  - The stuff-sets are equal if the stuff terms
```

```
        are equal, and the arity + stufftype terms of one are
        contained in the arity +stufftype term of the other.
        */

         stuff_set_eq((TE,3,SAE),A, B, yes) :-
             ProcNo = (TE,3,SAE),
             stuff(A,SA),stuff(B,SB),
        arity(A,ArA),arity(B,ArB),
        stuff_type(A,StA),stuff_type(B,StB),

        term_eq(ProcNo,SA,SB,yes),
        append(ArA,StA,TotalA),
        append(ArB,StB,TotalB),
             contained_in_check(ProcNo,TotalA,TotalB,yes).

/*   Set 2/Rule 4 -- The stuff-sets are equal if
             the stuff + arity + stufftype terms of one are
        contained in the stuff + arity +stufftype term of the other.
        */

         stuff_set_eq((TE,4,SAE),A, B, yes) :-
             ProcNo = (TE,4,SAE),
             stuff(A,SA),stuff(B,SB),
        arity(A,ArA),arity(B,ArB),
        stuff_type(A,StA),stuff_type(B,StB),

        append(ArA,[SA],SubTotalA),
        append(ArB,[SB],SubTotalB),

        append(SubTotalA,StA,TotalA),
        append(SubTotalB,StB,TotalB),
             contained_in_check(ProcNo,TotalA,TotalB,yes).

/* STUFF-attribute-SET EQUIVALENCE: equivalence of stuff-attribute-sets. */
/* stuff_attribute_set_eq(ProcNo,E1,E2,Answer). */
/* stuff_attribute-set(A) =~ stuff_attribute-set(B)
  if stuff_attribute_set_eq(ProcNo,A,B,yes) . */

/*   Set 3/Rule 1 -- The stuff_attribute-sets are equal if the stuff_attribute terms
        are equal, and the
        stuff_attribute_type terms are equal.
        */

         stuff_attribute_set_eq((TE,SE,1),A, B, yes) :-
             ProcNo = (TE,SE,1),
        stuff_attribute(A,SaA),stuff_attribute(B,SaB),
        stuff_attribute_type(A,SatA),stuff_attribute_type(B,SatB),

        term_eq(ProcNo,SaA,SaB,yes),
             term_eq(ProcNo,SatA,SatB,yes).

/*   Set 3/Rule 2 -- The stuff_attribute-sets are equal if
              the stuff_attribute terms are equal,
        and if the stuff_attribute type term of one is contained
        in the stuff_attribute type term of the other.
        */

         stuff_attribute_set_eq((TE,SE,2),A, B, yes) :-
             ProcNo = (TE,SE,2),
             stuff_attribute(A,SaA),stuff_attribute(B,SaB),
        stuff_attribute_type(A,SatA),stuff_attribute_type(B,SatB),

        term_eq(ProcNo,SaA,SaB,yes),
             contained_in_check(ProcNo,SatA,SatB,yes).
```

```
/*   Set 3/Rule 3 -- The stuff_attribute-sets are equal if
             the stuff_attribute + stuff_attributetype terms of one are
        contained in the stuff_attribute +stuff_attributetype term of the other.
        */

         stuff_attribute_set_eq((TE,SE,3),A, B, yes) :-
             ProcNo = (TE,SE,3),
             stuff_attribute(A,SaA),stuff_attribute(B,SaB),
        stuff_attribute_type(A,SatA),stuff_attribute_type(B,SatB),

        append([SaA],SatA,TotalA),
        append([SaB],SatB,TotalB),
             contained_in_check(ProcNo,TotalA,TotalB,yes).


/* TERM EQUIVALENCE: Term Equivalence Rules:

format: term_eq(WhichRule,Term1,Term2).
succeeds when Term1 and Term 2 are equivalent
   under WhichRule. */



/* To take care of the case when Term1 and Term2 are lists. */
/* In such cases, see if all elements of Term1 are
"contained in" Term2, and vice versa.
  The predicate term_list_LtoR_eq takes care of the above. */

         term_eq(ProcNo,[H1|T1],[H2|T2],yes) :-
            term_list_LtoR_eq(ProcNo,[H1|T1],[H2|T2],yes),
        term_list_LtoR_eq(ProcNo,[H2|T2],[H1|T1],yes).

    term_list_LtoR_eq(ProcNo,[],List2,yes).
    term_list_LtoR_eq(ProcNo,[First1|Rest1],List2,yes) :-
         contained_in(ProcNo,[First1],List2),
         term_eq(ProcNo,Resc1,List2,yes).

/* Set 1/Rule 1 -- Terms are equal if they match syntactically */

         term_eq((1,_,_), A, B,yes) :-
             A = B.

/* Set 1/Rule 2 Terms are equal if Rule 1 is true
    or if A and B are synonyms */

         term_eq((2,X,Y), A, B,yes) :-
        term_eq((1,X,Y), A, B,yes);
                  synonym(A, B).

/* Set 1/Rule 3 -- Terms are equal if Rule 1 or
   Rule 2 are true, or if A and B are related,
   i.e., they are contained in the same inheritance hierarchy. */

         term_eq((3,X,Y), A, B,yes) :-
        term_eq((1,X,Y), A, B,yes);
             term_eq((2,X,Y), A, B,yes);
        is_a(A,B);
             is_a(B,A).

/* If none of these work, then (A,B) are not equivalent. */
         term_eq(_,A,B,no(A,B)) :-
       !, fail.
```

```
/*  UTILITIES */

/* contained_in_check(Set1,Set2,SuperSet).
   succeeds when Set1 is contained in Set2,
   SuperSet indicates which one is the larger. */

contained_in_check(ProcNo,Set1,Set2,yes) :-
   contained_in(ProcNo,Set1,Set2).
contained_in_check(ProcNo,Set1,Set2,yes) :-
   contained_in(ProcNo,Set2,Set1).

/* contained_in(Set1,Set2).
   succeeds when Set1 is contained in Set2. */

/* empty list is contained in AnySet. */
contained_in(ProcNo,[],AnySet).

/* A set containing only 1 Member is contained in Set2
if Member is a member of Set2. */
contained_in(ProcNo,[Member],[First|Rest]) :-
    term_eq(ProcNo,Member,First,yes);
    not(Rest = []),
    contained_in(ProcNo,[Member],Rest).

/* A set containing a First member and the Rest of the set,
is contained in Set2 if Set2 contains the First member
as well as the Rest of the set. */
contained_in(ProcNo,[First|Rest],Set2) :-
    not(Rest = []),
    contained_in(ProcNo,[First],Set2),
    contained_in(ProcNo,Rest,Set2).


     printreport(ProcNo,(DB1,Sub1),(DB2,Sub2)) :-
        write('Please enter the name of the output file: '),
   read(FileName),
        tell(FileName),
   write('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%'),nl,
   printlist(['Results for procedure: ',ProcNo,' applied to ',
         [DB1,Sub1],' and ',[DB2,Sub2],'.']),
   nl,
   write('List of matches: '),
   setof0((E1,E2),tomato(ProcNo,[DB1,Sub1,E1],[DB2,Sub2,E2],match),MatchList),
   printlist(MatchList),
   nl,nl,
   write('List of homonyms: '),
   setof0((E1,E2),tomato(ProcNo,[DB1,Sub1,E1],[DB2,Sub2,E2],homonym),HomList),
   printlist(HomList),
   nl,nl,
   write('List of synonyms: '),
   setof0((E1,E2),tomato(ProcNo,[DB1,Sub1,E1],[DB2,Sub2,E2],synonym),SynList),
   printlist(SynList),
/*
   nl,nl,
   write('List of garbage: '),
   setof0((E1,E2),tomato(ProcNo,[DB1,Sub1,E1],[DB2,Sub2,E2],relax),Garbage),
   printlist(Garbage),
*/
   told.


    printlist([]).
    printlist([H|T]) :-
```

```
       nl,
       write(H),
       printlist(T).

  setof0(X,Y,Z) :- setof(X,Y,Z),!.
  setof0(_,_,[]) :- !.

/* Assume that synonyms are declared using the predicate synonyms/2.
    For example, synonyms(cost,price).  */

    synonym(A,B) :-
        synonyms(A,B).
    synonym(A,B) :-
        synonyms(B,A).
/*    synonym(A,B) :-
        synonyms(A,C),
        synonym(C,B).
    synonym(A,B) :-
        synonyms(C,A),
        synonym(C,B).     */

/* Multi-level classification hierarchies */
/* Assume that if A is_a B, there is a predicate isA(A,B) */

    is_a(A,B) :-
       isA(A,B).
    is_a(A,B) :-
       isA(A,C),
       is_a(C,B).


/* Retrieval of quiddity terms. */

stuff([DB,Subject,Element],Stuff) :-
    quiddity(DB,Subject,Element,Stuff,_,_,_,_).
arity([DB,Subject,Element],Arity) :-
    quiddity(DB,Subject,Element,_,Arity,_,_,_).
stuff_type([DB,Subject,Element],StuffType) :-
    quiddity(DB,Subject,Element,_,_,StuffType,_,_).
stuff_attribute([DB,Subject,Element],StuffAttribute) :-
    quiddity(DB,Subject,Element,_,_,_,StuffAttribute,_).
stuff_attribute_type([DB,Subject,Element],StuffAttributeType) :-
    quiddity(DB,Subject,Element,_,_,_,_,StuffAttributeType).
```

```
Page:    1 calvin:Renae-program:mg:mgbt222

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Results for procedure: 222

Applied to NAC Subject A and Craft Subject 2
.
(*First data element listed is from NAC and second is from CRAFT)


List of matches:

   NONE


List of homonyms:

firstname,firstname
lastname,lastname
section,section



List of synonyms:

crs,crs_name
crs,crs_number
dpt,dept
emph_area,emph
emph_area,emph_name
length,coreqtr
preq_crs,crs_name
preq_crs,crs_number
preq_crs,pre_req_num
preq_dpt,dept
preq_dpt,pre_req_dept
req_crs,crs_name
req_crs,crs_number
section,coreqtr
```

Primary Experiment Data, Synonym List (Vocabulary), and Classification Network

```
quiddity(nac,nr,crs,course,[],[nps],designator,[]).
quiddity(nac,nr,curr_ofcr,manager,[curriculum],[military_officer],surname,[]).
quiddity(nac,nr,curric_nam,curriculum,[],[nps],title,[]).
quiddity(nac,nr,degree_tit,degree,[],[nps],title,[]).
quiddity(nac,nr,dpt,department,[],[nps],identifier,[]).
quiddity(nac,nr,firstname,person,[],[],name,[given]).
quiddity(nac,nr,lastname,person,[],[nps],surname,[]).
quiddity(nac,nr,section,time_period,[],[course],designator,[]).
quiddity(nac,nr,hours,course,[],[],credit,[]).
quiddity(nac,nr,prof_phone,professor,[],[nps],telephone_number,office).
quiddity(nac,nr,emph_area,curriculum,[],[emphasis_area],credits,[required]).
quiddity(nac,nr,length,curriculum,[],[],term,[required]).
quiddity(nac,nr,preq_crs,course,[],[prerequisite,required],designator,[]).
quiddity(nac,nr,preq_dpt,department,[],[prerequisite,required],identifier,[]).
quiddity(nac,nr,req_crs,emphasis_area,[],[course,required],designator,[]).
quiddity(nac,rg,crs,course,[],[nps],identifier,[]).
quiddity(nac,rg,curr_ofcr,person,[],[military_officer],name,[surname]).
quiddity(nac,rg,curric_nam,curriculum,[curriculum],[],name,[]).
quiddity(nac,rg,degree_tit,degree,[],[nps],title,[]).
quiddity(nac,rg,dpt,department,[],[nps],name,[]).
quiddity(nac,rg,firstname,person,[],[],name,[given]).
quiddity(nac,rg,lastname,person,[],[],name,[surname]).
quiddity(nac,rg,section,time_period,[],[course],identifier,[]).
quiddity(nac,rg,hours,course,[],[],credits,[]).
quiddity(nac,rg,prof_phone,professor,[],[],telephone_number,[office]).
quiddity(nac,rg,emph_area,emphasis_area,[curriculum],[curriculum],name,[]).
quiddity(nac,rg,length,time_period,[],[curriculum],term,[]).
quiddity(nac,rg,preq_crs,course,[],[prerequisite],designator,[]).
quiddity(nac,rg,preq_dpt,department,[],[course],designator,[]).
quiddity(nac,rg,req_crs,course,[],[emphasis_area],designator,[required]).
quiddity(nac,mg,crs,course,[],[nps],designator,[]).
quiddity(nac,mg,curr_ofcr,manager,[curriculum],[military_officer],surname,[]).
quiddity(nac,mg,curric_nam,curriculum,[],[nps],title,[]).
quiddity(nac,mg,degree_tit,degree,[],[],title,[]).
quiddity(nac,mg,dpt,department,[],[nps],identifier,[]).
quiddity(nac,mg,firstname,person,[],[nps],name,[]).
quiddity(nac,mg,lastname,person,[],[nps],surname,[]).
quiddity(nac,mg,section,time_period,[],[class],identifier,[]).
quiddity(nac,mg,hours,course,[],[],credits,[]).
quiddity(nac,mg,prof_phone,professor,[],[nps],telephone_number,[]).
quiddity(nac,mg,emph_area,emphasis_area,[],[],title,[]).
quiddity(nac,mg,length,time_period,[],[curriculum],designator,[]).
quiddity(nac,mg,preq_crs,course,[],[prerequisite,nps],designator,[]).
quiddity(nac,mg,preq_dpt,department,[],[prerequisite,nps],identifier,[]).
quiddity(nac,mg,req_crs,course,[],[required,nps],designator,[]).
quiddity(craft,jc,ssn,student,[],[],identifier,[social_security]).
quiddity(craft,jc,lastname,student,[],[],surname,[]).
quiddity(craft,jc,firstname,student,[],[],name,[given]).
quiddity(craft,jc,section,section,[],[nps],designator,[]).
quiddity(craft,jc,dept,department,[],[nps],identifier,[]).
quiddity(craft,jc,crs_number,course,[],[nps],identifier,[]).
quiddity(craft,jc,crs_name,course,[],[nps],name,[]).
quiddity(craft,jc,emph,emphasis_area,[],[nps],identifier,[]).
quiddity(craft,jc,emph_name,emphasis_area,[],[nps],name,[]).
quiddity(craft,jc,pre_req_dept,department,[],[prerequisite,nps],identifier,[]).
quiddity(craft,jc,pre_req_num,course,[],[prerequisite,nps],identifier,[]).
quiddity(craft,jc,qtr_name,quarter,[],[],name,[]).
quiddity(craft,jc,qtr,quarter,[],[],identifier,[]).
quiddity(craft,jc,yr,year,[],[],identifier,[]).
quiddity(craft,jc,coreqtr,quarter,[student],[current],identifier,[]).
quiddity(craft,js,ssn,student,[],[],identifier,[social_security]).
quiddity(craft,js,lastname,person,[],[],surname,[]).
quiddity(craft,js,firstname,person,[],[],name,[given]).
quiddity(craft,js,section,section,[],[curriculum],identifier,[]).
quiddity(craft,js,dept,department,[],[nps],designator,[]).
quiddity(craft,js,crs_number,course,[],[nps],designator,[]).
quiddity(craft,js,crs_name,course,[],[nps],name,[]).
quiddity(craft,js,emph,emphasis_area,[],[nps],identifier,[]).
quiddity(craft,js,emph_name,emphasis_area,[],[nps],name,[]).
quiddity(craft,js,pre_req_dept,department,[],[prerequisite,nps],designator,[]).
quiddity(craft,js,pre_req_num,course,[],[prerequisite,nps],designator,[]).
quiddity(craft,js,qtr_name,quarter,[],[],name,[]).
quiddity(craft,js,qtr,quarter,[],[],identifier,[]).
quiddity(craft,js,yr,time_period,[course],[],year,[given]).
quiddity(craft,js,coreqtr,quarter,[time],[current],indicator,[]).
quiddity(craft,bt,ssn,student,[],[military_officer],identifier,[social_security]).
quiddity(craft,bt,lastname,student,[],[military_officer],surname,[]).
quiddity(craft,bt,firstname,student,[],[military_officer],name,[]).
quiddity(craft,bt,section,section,[student,curriculum],[],identifier,[]).
quiddity(craft,bt,dept,department,[],[nps],identifier,[]).
quiddity(craft,bt,crs_number,course,[],[],identifier,[]).
quiddity(craft,bt,crs_name,course,[],[],name,[]).
quiddity(craft,bt,emph,emphasis_area,[],[curriculum],designator,[]).
quiddity(craft,bt,emph_name,emphasis_area,[],[curriculum],name,[]).
quiddity(craft,bt,pre_req_dept,department,[],[prerequisite],designator,[]).
quiddity(craft,bt,pre_req_num,course,[],[prerequisite],identifier,[]).
quiddity(craft,bt,qtr_name,time_period,[],[quarter],name,[]).
quiddity(craft,bt,qtr,time_period,[],[],quarter,[]).
quiddity(craft,bt,yr,time_period,[],[],year,[]).
quiddity(craft,bt,coreqtr,quarter,[curriculum,time],[],identifier,[]).
```

```
synonym(course,class).
synonym(designator,identifier).
synonym(designator,name).
synonym(designator,title).
synonym(quarter,term).
synonym(quarter,time_period).

isA(professor,person).
isA(manager,person).
isA(student,person).
isA(military_officer,person).
isA(prerequisite,required).
isA(NPS,university).
isA(current,time).
isA(quarter,term).
isA(year,term).
isA(title,name).
isA(surname,name).
isA(curriculum,department).
isA(course,curriculum).
isA(emphasis_area,curriculum).
isA(class,course).
isA(section,class).
```

```
Page:    1 masterDATA.DOS

quiddity(nac,rb,crs,course,[],[nps],identifier,[]).
quiddity(nac,rb,curr_ofcr,manager,[],[curriculum,military_officer],name,[]).
quiddity(nac,rb,curric_nam,curriculum,[],[nps],title,[]).
quiddity(nac,rb,degree_tit,degree,[],[nps],title,[]).
quiddity(nac,rb,dpt,department,[],[nps],designator,[]).
quiddity(nac,rb,firstname,person,[],[nps],name,[given]).
quiddity(nac,rb,lastname,person,[],[nps],surname,[]).
quiddity(nac,rb,section,time_period,[],[course],identifier,[]).
quiddity(nac,rb,hours,course,[],[],credits,[]).
quiddity(nac,rb,prof_phone,professor,[],[nps],telephone_number,[office]).
quiddity(nac,rb,emph_area,emphasis_area,[],[curriculum],title,[]).
quiddity(nac,rb,length,completion,[curriculum],[],term,[required]).
quiddity(nac,rb,preq_crs,course,[],[nps,prerequisite],identifier,[]).
quiddity(nac,rb,preq_dpt,department,[],[prerequisite],identifier,[]).
quiddity(nac,rb,req_crs,course,[],[emphasis_area,required],identifier,[]).
quiddity(craft,rb,ssn,student,[],[],identifier,[social_security]).
quiddity(craft,rb,lastname,student,[],[],surname,[]).
quiddity(craft,rb,firstname,student,[],[],name,[given]).
quiddity(craft,rb,section,section,[],[class],identifier,[]).
quiddity(craft,rb,dept,department,[],[nps],identifier,[]).
quiddity(craft,rb,crs_number,course,[],[nps],identifier,[]).
quiddity(craft,rb,crs_name,course,[],[nps],title,[]).
quiddity(craft,rb,emph,emphasis_area,[],[nps],identifier,[]).
quiddity(craft,rb,emph_name,emphasis_area,[],[nps],title,[]).
quiddity(craft,rb,pre_req_dept,department,[],[prerequisite,nps],identifier,[]).
quiddity(craft,rb,pre_req_num,course,[],[prerequisite,nps],identifier,[]).
quiddity(craft,rb,qtr_name,quarter,[],[],name,[]).
quiddity(craft,rb,qtr,quarter,[],[],identifier,[]).
quiddity(craft,rb,yr,completion,[course,student],[],year,[]).
quiddity(craft,rb,coreqtr,student,[time],[],quarter,[]).

synonyms(course,class).
synonyms(designator,identifier).
synonyms(designator,name).
synonyms(designator,title).
synonyms(quarter,term).
synonyms(quarter,time_period).
synonyms(title,name).
synonyms(title, identifier).
synonyms(name,identifier).
synonyms(quarter,section).
synonyms(term,time_period).

isA(professor,person).
isA(manager,person).
isA(student,person).
isA(military_officer,person).
isA(prerequisite,required).
isA(nps,university).
isA(current,time).
isA(quarter,term).
isA(year,term).
isA(title,name).
isA(surname,name).
isA(curriculum,department).
isA(course,curriculum).
isA(emphasis_area,curriculum).
isA(class,course).
isA(section,class).
```

APPENDIX C -- PRIMARY EXPERIMENT

This Appendix contains samples of the contents of the packet given to the students during the primary experiment. Additionally, the actual experiment data results along with the master quiddities is provided. Finally, detailed tabular experiment data (from prototype) is included along with several bar graphs for further clarification. Items specified above are located on the following pages:

EXPERIMENT #2

A. PURPOSE

The purpose of this experiment is to gather data to assist me in analyzing the concept of "quiddity". The first experiment gave everyone a broad view of the quiddity concept and practice in applying the concept to a database. The second experiment is the more important of the two and will be more formally structured. I am still interested in any and all comments you may have regarding quiddity.

B. REVIEW OF THE QUIDDITY CONCEPT

"Quiddity" is the name given to the description of what information is captured by the data element. We are attempting to capture the "meaning" of what the data element represents.

1. Components of Quiddity

a. Quiddity is made up of five components, **stuff**, **stuff type**, **stuff attribute**, **stuff attribute type**, and **arity**. To find values for these components, we must answer the following questions.

* **STUFF**- What is it about?

STUFF TYPE- What sort of stuff is it?

* **STUFF ATTRIBUTE**- What is it about the stuff you are interested in?

STUFF ATTRIBUTE TYPE- What sort of stuff attribute is it?

ARITY- What is the stuff a function of?

b. Some important "rules of thumb" to follow are:

- **Most important fields are STUFF and STUFF ATTRIBUTE.** You must have both of these to have a meaningful quiddity, just like you must have a subject and a verb to have a complete sentence. **There is one and only one value for these components in each quiddity expression!**
- Capture "meaning" of what the data element represents.
- When determining quiddity, look at the definition of the data elements, rather than the names of the data elements themselves.
- Some data element names are deceptive/un-informative

2. New Approach

As stated earlier, the two most important components of quiddity are stuff and stuff attribute. If we can find these, we will have captured the data element meaning. Most people seem to have difficulty distinguishing between the two components. I have developed some new questions to ask yourself when defining these components. I hope these questions together with the above method will clarify the concept.

a. The idea is to find the stuff attribute first. Once this is done, the stuff component follows naturally. Try these steps:

- Look at a collection of actual data contained in the field

- Classify the data by grouping the collection under a general heading or name which answers the question "What is it?" or "What are these?" What do you actually see in the field? We want to categorize the actual words, codes, numbers, etc., that we see in the field. **The data is a MEASURE of something. The MEASURE is the stuff attribute and the SOMETHING is the stuff!** We are not concerned with what the data are representative of in the physical or concrete sense. We are looking for an abstract noun. Stuff attribute is not a dimension!!

b. Some examples are:

The data in the field looks like this "$23.34". This is the **COST** (stuff attribute) of **SOMETHING** (stuff).

The data in the field looks like this "sofa", "chair", "TV", "table", etc. You might be tempted to say that the "category" of this data is "furniture" but you would be wrong! We want to capture a measure of the data, not what the data represents in the physical sense. What measure is this, or what are they? They are **NAMES**! Names of what? Property! So, the stuff attribute is "name" and the stuff is "property". The questions above are also answered!

(Note: You will still have the data dictionary to look at too!)

WORK SHEET[1]

(Updated for Experiment #2)

1. How do we capture the "meaning" of what a data element represents?

a. A proposed method for capturing this "meaning" uses a type of formal "language" with various rules for forming the definition of the "meaning." This definition or description is called "quiddity."

> "From the *Oxford English Dictionary*, quiddity is 'The real nature or essence of a thing; that which makes a thing what it is.' Of course, ... [the proposed] language for expressing quiddities is only a model, or approximation, of genuine quiddity, if it exists."

Example 1:

- DATABASE 1
  - Variable: *purchase_cost*
  - Description:

    "Purchase cost of a truck"

- DATABASE 2
  - Variable: *cost_of_purchase*
  - Description:

    "Cost of purchase of a truck"

b. Let's begin defining the basic component of quiddity.

Example 1a:

- DATABASE 1
  - Variable: *purchase_cost*
  - Description:

    "Purchase cost of a truck"
  - Dimension: currency
  - Stuff: truck

- DATABASE 2
  - Variable: *cost_of_purchase*
  - Description:

    "Cost of purchase of a truck"
  - Dimension: currency
  - Stuff: truck

[1] All examples and quotes in this work sheet have been borrowed from the following reference: Bhargava, Hemant K., Steven O. Kimbrough, and Ramayya Krishnan, *Unique Names Violations: A Problem for Model Integration or You Say Tomato, I Say Tomahto* (University of Pennsylvania, Department of Decision Sciences, Working Paper, 1990, forthcoming, ORSA Journal on Computing, Spring 1991), pp. 5-8.

2. Now, let's change the variables slightly.

a. Notice that the description changed but not the "dimension" or "stuff."

Example 2:

- DATABASE 1
  - Variable: *purchase_cost*
  - Description:
    "Cost of purchasing a truck"
  - Dimension: currency
  - Stuff: truck

- DATABASE 2
  - Variable: *production_cost*
  - Description:
    "Cost of producing a truck"
  - Dimension: currency
  - Stuff: truck

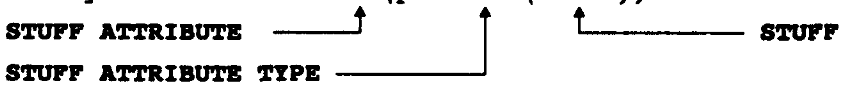
b. What is the *quiddity*?

Sample line of reasoning used in Example 2a to describe "quiddity."

> Both variables are about the same stuff: trucks. They differ in what it is they represent *about* trucks. What is it about trucks they describe? Cost, in both cases. What kind or sort of cost are we interested in? Purchasing in one case and production in the other. This line of reasoning suggests the quiddity descriptions in Example 2a.

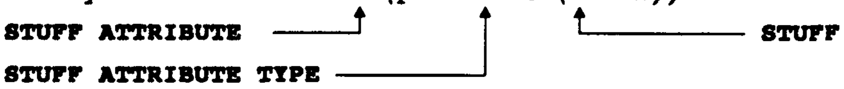
Example 2a:

- DATABASE 1
  - Variable: *purchase_cost*
  - Description: "Cost of purchasing a truck"
  - Dimension: currency
  - Quiddity: *cost(purchase(truck))*



  - Quiddity Paraphrase: "the purchase cost of a truck"

- DATABASE 2
  - Variable: *production_cost*
  - Description: "Cost of producing a truck"
  - Dimension: currency
  - Quiddity: *cost(production(truck))*



  - Quiddity Paraphrase: "the production cost of a truck"

3. Below is a list of several data elements in a "Home Inventory" database. See if you can define the quiddity for each data element.

DATA DICTIONARY EXCERPT:

FIELD NAME	TYPE	DESCRIPTION
ITEM	1{character}40	Identifies a specific piece of property, i.e., sofa, dining room chair, TV, etc.
QUANTITY	1{integer}3	Identifies the total number of like items or pieces of property owned, i.e., "2" if two sofas are owned.
VALUE	1{integer}8	Identifies the current replacement cost of a specific piece of property.
DATE	1{date}8	The month, day, and year the property was purchased or acquired.
PRICE	1{integer}8	Identifies the amount paid for a specific piece of property.
WEIGHT	1{integer}5	The total number of pounds a specific piece of property weighs.
FREE_WEIGHT	1{logical}1	Whether weight of a specific piece of property applies toward the professionalweight allowance or not, i.e., "Y" if yes or "N" if no.

DATA ELEMENTS	QUIDDITY				
	STUFF	ARITY ARGUMENTS	STUFF TYPE	STUFF ATTRIBUTE	STUFF ATTRIBUTE TYPE
ITEM	item		household		
QUANTITY	item		household like	quantity	
VALUE	property	time	private	cost	replacement
DATE	item		household	date	purchase
PRICE	item		household	cost	purchase
WEIGHT	item		household	weight	
FREE_WEIGHT	category		professional property	indicator	

EXPERIMENT #2

Instructions:

1. Determine the quiddity for each data element listed. Record the components of the quiddity in the appropriate columns of the row listing the data element. Please write <u>legibly</u>.

2. Please keep track of the order in which you determine the quiddity components for each data element by placing a number in the upper left corner of the appropriate "box" in the table. For example, if the first term you define for the first data element is its stuff, the second term is its stuff type, and the third term is its stuff attribute, the table would look like this:

DATA ELEMENT	STUFF	ARITY	STUFF TYPE	STUFF ATTRIBUTE	STUFF ATTRIBUTE TYPE
PRICE	2 item		3 household	1 cost	4 retail

3. When defining quiddities, *you must have **exactly ONE "stuff" component term** and **exactly ONE "stuff attribute" component term**.* However, the components **ARITY, STUFF TYPE,** and **STUFF ATTRIBUTE TYPE** may be left blank or have one or more terms for each quiddity, depending on the definition you are writing. If there is more than one term, list them together in the appropriate "box" and place each term's ordering number to its left in the box.

4. I am interested in the "method" you use in determining the quiddity, particularly in the "thought process" you go through in working through this experiment. Please jot down the method you found most helpful in determining the quiddities. Any comments or suggestions you have (even in bullet form) is appreciated.

COMMENTS:

NAC DATABASE

DATA ELEMENTS	QUIDDITY				
	STUFF	ARITY ARGUMENTS	STUFF TYPE	STUFF ATTRIBUTE	STUFF ATTRIBUTE TYPE
CRS					
CURR_OFCR					
CURRIC_NAM					
DEGREE_TIT					
DPT					
FIRSTNAME					
LASTNAME					
SECTION					
HOURS					
PROF_PHONE					
EMPH_AREA					
LENGTH					
PREQ_CRS					
PREQ_DPT					
REQ_CRS					

CRAFT DATABASE

DATA ELEMENTS	QUIDDITY				
	STUFF	ARITY ARGUMENTS	STUFF TYPE	STUFF ATTRIBUTE	STUFF ATTRIBUTE TYPE
SSN					
LASTNAME					
FIRSTNAME					
SECTION					
DEPT					
CRS_NUMBER					
CRS_NAME					
EMPH					
EMPH_NAME					
PRE_REQ_DEPT					
PRE_REQ_NUM					
QTR_NAME					
QTR					
YR					
COREQTR					

VOCABULARY

STUFF:

completion
course
curriculum
degree
department
emphasis_area
manager
person
professor
quarter
section
student
time_period

STUFF TYPES:

class
course
current
curriculum
emphasis_area
military_officer
NPS
prerequisite
required
university

STUFF ATTRIBUTES:

credits
designator
identifier
name
quarter
surname
telephone_number
term
title
year

ARITY ARGUMENTS:

course
curriculum
student
time

STUFF ATTRIBUTE TYPES:

given
office
required
social_security

NAC DATABASE DATA DICTIONARY

FIELD NAME	TYPEDESCRIPTION	
CRS	1{DIGIT}4	Four digit number assigned to a particular course of instruction at the Naval Postgraduate School.
CURR_OFCR	STRING	Military officer assigned to manage a particular curriculum.
CURRIC_NAM	STRING	Title of curriculum course of study.
DEGREE_TIT	STRING	Title of Degree which can be awarded by the Naval Postgraduate School.
DPT	1{character}2	Two letter code that represents a particular department of the Naval Postgraduate School.
EMPH_AREA	STRING	Name of an emphasis area of study that students may elect courses from as a sub-specialty area within a particular curriculum.
FIRSTNAME	STRING	Person's given name
HOURS	1{DIGIT}1	Credit assigned to each course of instruction that meets graduation and degree requirements of so many credit hours.
LASTNAME	STRING	Surname of any person at Naval Postgraduate School.
LENGTH	1{DIGIT}2	Length of time (in months) required to complete course of study in a particular curriculum.
PREQ_CRS	1{DIGIT}4	Course number that when combined with Prerequisite Department, identifies a course that meets the requirements of another course.
PREQ_DPT	1{STRING}2	2 letter code for a prerequisite department.
PROF_PHONE	+ 1{DIGIT}3 + + 1{DIGIT)3 + 1{DIGIT}4	Telephone number of a particular professor's office.
REQ_CRS	1{DIGIT}4	Four digit number representing courses that are required for a particular emphasis area.
SECTION	1{DIGIT}1	Time period in which a given course is taught.

CRAFT DATABASE DATA DICTIONARY

FIELD NAME	TYPE	DESCRIPTION
SSN	000..999 + '-' 00..99 + '-' 0000..9999	Identifies students uniquely with their social security numbers.
LASTNAME	1{CHARACTER}15	Identifies the last name of an individual.
FIRSTNAME	1{CHARACTER}15	Identifies the first name of an individual.
SECT	1{CHARACTER}4	Identifies the section within a curriculum that a student is assigned to.
DEPT	1{CHARACTER}2	The unique two letter code which identifies an Academic Department at the Naval Postgraduate School.
CRS_NUMBER	1{DIGIT}4	Four digit number assigned to a particular course of instruction at the Naval Postgraduate School.
CRS_NAME	1{CHARACTER}25	Name assigned to a particular course of study at the Naval Postgraduate School.
EMPH	1{CHARACTER}3	Three letter code identifying an emphasis area course of study at the Naval Postgraduate School.
EMPH_NAME	1{CHARACTER}25	Name assigned to a particular emphasis area course of study at the Naval Postgraduate School.
QTR_NAME	1{CHARACTER}6	Name given to each academic quarter of the school year, i.e., Fall, Winter, Spring, and Summer.
QTR	1{DIGIT}2	Two digit code identifying a particular academic quarter of the school year, i.e., 01=Fall, 02=Winter, 03=Spring, and 04=Summer.
YR	1{DIGIT}2	Two digit code identifying a particular year a student completed a course, i.e., 90=1990, 91=1991, etc.
PRE_REQ_DEPT	1{CHARACTER}2	Two letter code identifying a Prerequisite Department within the Naval Postgraduate School.
PRE_REQ_NUM	1{DIGIT}4	Four digit number assigned to a particular prerequisite course of instruction at the Naval Postgraduate School.
COREQTR	1{DIGIT}2	Two digit code which indicates which quarter of the curriculum a student is currently in, i.e., 01, 02, 03, 04, 05, and 06.

__Output Reports__ The predefined functions of the NAC are provided via use of established queries and designed reports which can be accessed from the dBase IV Control Center. These reports and their associated functions are listed in the following table:

	Predefined Function
AVAL_367	Available Courses for Curriculum 367 which are offered in the Summer Quarter
PREREQ	List of Prerequisite Courses for a specified Course or group of Courses
THESIS	Thesis Support and Special Interests for a specified Professor
DEG_INFO	Information related to a specified Degree Program
CRS_TAUT	List of Courses Taught by a specified Professor
EMPH_CSE	Requirements for Specified Emphasis Area

Examples of these reports have been attached.

SAMPLE NAC DATABASE REPORTS

AVAL_367

Curriculum 367 Courses Available for Summer Quarter

Course	Section	Professor	Description
CS3010	1	Stevens	Computing Devices and Systems
1S3170	1	Haga	Economic Evaluation of IS
1S4185	1	Bui	Decision Support Systems

CRS_TAUT

Courses Taught by: Bui, Tung

Office: 1320 Phone: (408) 646-3260

		Quarter/s Offered			
Course	Description	W	Sp	Su	F
IS4200	System Analysis and Design	N	Y	N	Y
IS4185	Decision Support Systems	Y	N	Y	N

DEG_INFO

Degrees Offered by Curriculum: 367

Degree Title: MS in Information System Management

Curriculum: Computer Systems Management

APC: 335 P-Code: 0095P Length: 18 Months

Convenes: Winter-N, SpringY, Summer-N, Fall-Y

SAMPLE NAC DATABASE REPORTS

EMPH_CSE

Emphasis Requirements for: IRM

Required Courses	Option Course	Elective Courses
IS3220	IS4184	CS4601
IS3220	IS4184	IS3000

PREREQ

Course Prerequisites

Course	Prereq Course	Option Course	Remarks
MN4154	MN2155	MN3161	
IS3503	IS3502		Can be concurrent

THESIS

Specialized Data for: Haga, William

Office: 1218 Phone: (408) 646-3094

Sabbaticals: NONE

Special Duties: Adjunct Professor of Management Information Systems, Naval Postgraduate School.

Special Interest: Studying the research methods used to gauge the success of information systems. He is interested in the by-products of systems implementations on small groups. His other research include the relationship of organizational structure and culture to information system success.

Published works: Academy of management review, Accounting Reviews, American Journal of Economics and Sociology, American Sociological review, Astronautics and aeronautics, Behavioral Science, Computers and Security, Data Processing and Communication Security, Journal of Contemporary Sociology, Journal of the System Safety Society and Organizational Behavior and Human Performance.

SAMPLE CRAFT DATABASE REPORTS

```
Page No. 1              STUDENT
05/19/90

SSN    LASTNAME        FIRSTNAME     SECTION   EMPH    COREQTR

223747355              APPLE          PAUL       PLO1      DSS        01
270646400              GREEN          SALLY      PL01      NET        01
555229999              JONES          BILL       PL01      DSS        02
222774444              MARS           BRYON      PL03      TAG        06
111884422              FRANTZ         JOAN       PL03      IRM        04
```

```
Page No 1           EMPHASIS

DEPT     CRS_NUMBER    PRE_REQ_DEPT    PRE_REQ_NUM      CRS_NAME

CS           2972          CS             2970        ADA FROM THE BEGINNING
CS           3010          CS             2970        ADA FROM THE BEGINNING
CS           3030          CS             3010        SOFTWARE DEVELOPMENT
IS           2000
IS           2100
IS           3000          CS             3010
IS           3000          IS             3170
```

NAC DATABASE
Subject A

DATA ELEMENTS	QUIDDITY				
	STUFF	ARITY (ARGUMENTS)	STUFF TYPE	STUFF ATTRIBUTE	STUFF ATTRIBUTE TYPE
CRS	course		NPS	designator	
CURR_OFCR	manager	curriculum	military_ officer	surname	
CURRIC_NAM	curriculum		NPS	title	
DEGREE_TIT	degree			title	
DPT	department		NPS	identifier	
FIRSTNAME	person		NPS	name	
LASTNAME	person		NPS	surname	
SECTION	time_period		class	identifier	
HOURS	course			credits	
PROF_PHONE	professor		NPS	telephone_ number	
EMPH_AREA	emphasis_area			title	
LENGTH	time_period		curriculum	designator	
PREQ_CRS	course		prerequisite NPS	designator	
PREQ_DPT	department		prerequisite NPS	identifier	
REQ_CRS	course		required NPS	designator	

NAC DATABASE
Subject B

DATA ELEMENTS	QUIDDITY				
	STUFF	ARITY (ARGUMENTS)	STUFF TYPE	STUFF ATTRIBUTE	STUFF ATTRIBUTE TYPE
CRS	course		NPS	designator	
CURR_OFCR	manager	curriculum	military_ officer	surname	
CURRIC_NAM	curriculum		NPS	title	
DEGREE_TIT	degree		NPS	title	
DPT	department		NPS	identifier	
FIRSTNAME	person			name	given
LASTNAME	person		NPS	surname	
SECTION	time_period		course	designator	
HOURS	course			credits	
PROF_PHONE	professor		NPS	telephone_ number	office
EMPH_AREA	curriculum		emphasis_area	credits	required
LENGTH	curriculum			term	required
PREQ_CRS	course		prerequisite required	designator	
PREQ_DPT	department		prerequisite required	identifier	
REQ_CRS	emphasis_area		required course	designator	

NAC DATABASE
Subject C

DATA ELEMENTS	QUIDDITY				
	STUFF	ARITY (ARGUMENTS)	STUFF TYPE	STUFF ATTRIBUTE	STUFF ATTRIBUTE TYPE
CRS	course		NPS	identifier	
CURR_OFCR	person		military_ officer	name	surname
CURRIC_NAM	curriculum	curriculum		name	
DEGREE_TIT	degree		NPS	title	
DPT	department		NPS	name	
FIRSTNAME	person			name	given
LASTNAME	person			name	surname
SECTION	time_period		course	identifier	
HOURS	course			credits	
PROF_PHONE	professor			telephone_ number	office
EMPH_AREA	emphasis_area	curriculum	curriculum	name	
LENGTH	time_period		curriculum	term	
PREQ_CRS	course		prerequisite	designator	
PREQ_DPT	department		course	designator	
REQ_CRS	course		emphasis_area	designator	required

CRAFT DATABASE
Subject 1

DATA ELEMENTS	QUIDDITY				
	STUFF	ARITY ARGUMENTS	STUFF TYPE	STUFF ATTRIBUTE	STUFF ATTRIBUTE TYPE
SSN	student			identifier	social_ security
LASTNAME	student			surname	
FIRSTNAME	student			name	given
SECTION	section		NPS	designator	
DEPT	department		NPS	identifier	
CRS_NUMBER	course		NPS	identifier	
CRS_NAME	course		NPS	name	
EMPH	emphasis_area		NPS	identifier	
EMPH_NAME	emphasis_area		NPS	name	
PRE_REQ_DEPT	department		prerequisite NPS	identifier	
PRE_REQ_NUM	course		prerequisite NPS	identifier	
QTR_NAME	quarter			name	
QTR	quarter			identifier	
YR	year			identifier	
COREQTR	quarter	student	current	identifier	

CRAFT DATABASE
Subject 2

DATA ELEMENTS	QUIDDITY				
	STUFF	ARITY ARGUMENTS	STUFF TYPE	STUFF ATTRIBUTE	STUFF ATTRIBUTE TYPE
SSN	student		military_ officer	identifier	social_ security
LASTNAME	student		military_ officer	surname	
FIRSTNAME	student		military_ officer	name	
SECTION	section	student curriculum		identifier	
DEPT	department		NPS	identifier	
CRS_NUMBER	course			identifier	
CRS_NAME	course			name	
EMPH	emphasis_area		curriculum	designator	
EMPH_NAME	emphasis_area		curriculum	name	
PRE_REQ_DEPT	department		prerequisite	designator	
PRE_REQ_NUM	course		prerequisite	identifier	
QTR_NAME	time_period		quarter	name	
QTR	time_period			quarter	
YR	time_period			year	
COREQTR	quarter	curriculum time		identifier	

CRAFT DATABASE
Subject 3

DATA ELEMENTS	QUIDDITY				
	STUFF	ARITY (ARGUMENTS)	STUFF TYPE	STUFF ATTRIBUTE	STUFF ATTRIBUTE TYPE
SSN	student			identifier	social_ security
LASTNAME	person			surname	
FIRSTNAME	person			name	given
SECTION	section		curriculum	identifier	
DEPT	department		NPS	designator	
CRS_NUMBER	course		NPS	designator	
CRS_NAME	course		NPS	name	
EMPH	emphasis_area		NPS	identifier	
EMPH_NAME	emphasis_area		NPS	name	
PRE_REQ_DEPT	department		prerequisite NPS	designator	
PRE_REQ_NUM	course		prerequisite NPS	designator	
QTR_NAME	quarter			name	
QTR	quarter			identifier	
YR	time_period	course		year	given
COREQTR	quarter	time	current	indicator	

NAC DATABASE

MASTER

DATA ELEMENTS	QUIDDITY				
	STUFF	ARITY (ARGUMENTS)	STUFF TYPE	STUFF ATTRIBUTE	STUFF ATTRIBUTE TYPE
CRS	course		NPS	identifier	
CURR_OFCR	manager		military_officer curriculum	name	
CURRIC_NAM	curriculum		NPS	title	
DEGREE_TIT	degree		NPS	title	
DPT	department		NPS	designator	
FIRSTNAME	person		NPS	name	given
LASTNAME	person		NPS	surname	
SECTION	time_period		course	identifier	
HOURS	course			credits	
PROF_PHONE	professor		NPS	telephone_number	office
EMPH_AREA	emphasis_area		curriculum	title	
LENGTH	completion	curriculum		term	required
PREQ_CRS	course		NPS prerequisite	identifier	
PREQ_DPT	department		prerequisite	identifier	
REQ_CRS	course		required emphasis_area	identifier	

CRAFT DATABASE
MASTER

DATA ELEMENTS	QUIDDITY				
	STUFF	ARITY (ARGUMENTS)	STUFF TYPE	STUFF ATTRIBUTE	STUFF ATTRIBUTE TYPE
SSN	student			identifier	social_ security
LASTNAME	student			surname	
FIRSTNAME	student			name	given
SECTION	section		class	identifier	
DEPT	department		NPS	identifier	
CRS_NUMBER	course		NPS	identifier	
CRS_NAME	course		NPS	title	
EMPH	emphasis_area		NPS	identifier	
EMPH_NAME	emphasis_area		NPS	title	
PRE_REQ_DEPT	department		NPS prerequisite	identifier	
PRE_REQ_NUM	course		NPS prerequisite	identifier	
QTR_NAME	quarter			name	
QTR	quarter			identifier	
YR	completion	course student		year	
COREQTR	student	time		quarter	

NAC / CRAFT DATABASE COMPARISONS BY PROCEDURE (Master)

Synonyms -- Master

Test	# Found	Type I	Type II
111	1	0	4
122	5	2	2
132	5	2	2
143	5	2	2
211	3	1	3
222	11	7	1
232	11	7	1
243	11	7	1
311	13	11	3
322	31	27	1
332	31	27	1
343	44	39	0

Synonyms -- Master

Test	# Found	Type I	Type II
111	1	0	4
211	3	1	3
311	13	11	3
122	5	2	2
222	11	7	1
322	31	27	1
132	5	2	2
232	11	7	1
332	31	27	1
143	5	2	2
243	11	7	1
343	44	39	0

Homonyms -- Master

Test	# Found	Type I	Type II
111	3	0	0
122	3	0	0
132	3	0	0
143	3	0	0
211	3	0	0
222	3	0	0
232	3	0	0
243	3	0	0
311	3	0	0
322	1	0	2
332	1	0	2
343	0	0	3

Homonyms -- Master

Test	# Found	Type I	Type II
111	3	0	0
211	3	0	0
311	3	0	0
122	3	0	0
222	3	0	0
322	1	0	2
132	3	0	0
232	3	0	0
332	1	0	2
143	3	0	0
243	3	0	0
343	0	0	3

NAC / CRAFT DATABASE COMPARISONS BY PROCEDURE
(Average Totals of the 9 Comparisons for Each Procedure)

Synonyms -- Experiment

Test	# Found	Type I	Type II
111	1.0	0.0	4.0
211	2.9	0.7	2.8
311	10.4	8.2	2.8
122	2.3	1.0	3.7
222	11.1	7.4	1.3
322	35.4	31.3	0.9
132	2.3	1.0	3.7
232	10.8	7.1	1.3
332	31.9	27.8	0.9
143	2.7	1.3	3.7
243	13.0	9.3	1.3
343	46.1	41.4	0.3

Synonyms -- Experiment

Test	# Found	Type I	Type II
111	1.0	0.0	4.0
122	2.3	1.0	3.7
132	2.3	1.0	3.7
143	2.7	1.3	3.7
211	2.9	0.7	2.8
222	11.1	7.4	1.3
232	10.8	7.1	1.3
243	13.0	9.3	1.3
311	10.4	8.2	2.8
322	35.4	31.3	0.9
332	31.9	27.8	0.9
343	46.1	41.4	0.3

Homonyms -- Experiment

Test	# Found	Type I	Type II
111	2.8	0.0	0.2
211	2.8	0.0	0.2
311	2.6	0.0	0.4
122	2.4	0.0	0.6
222	2.4	0.0	0.6
322	1.3	0.0	1.7
132	2.4	0.0	0.6
232	2.4	0.0	0.6
332	1.3	0.0	1.7
143	2.3	0.0	0.7
243	2.3	0.0	0.7
343	0.7	0.0	2.3

Homonyns -- Experiment

Test	# Found	Type I	Type II
111	2.8	0.0	0.2
122	2.4	0.0	0.6
132	2.4	0.0	0.6
143	2.3	0.0	0.7
211	2.8	0.0	0.2
222	2.4	0.0	0.6
232	2.4	0.0	0.6
243	2.3	0.0	0.7
311	2.6	0.0	0.4
322	1.3	0.0	1.7
332	1.3	0.0	1.7
343	0.7	0.0	2.3

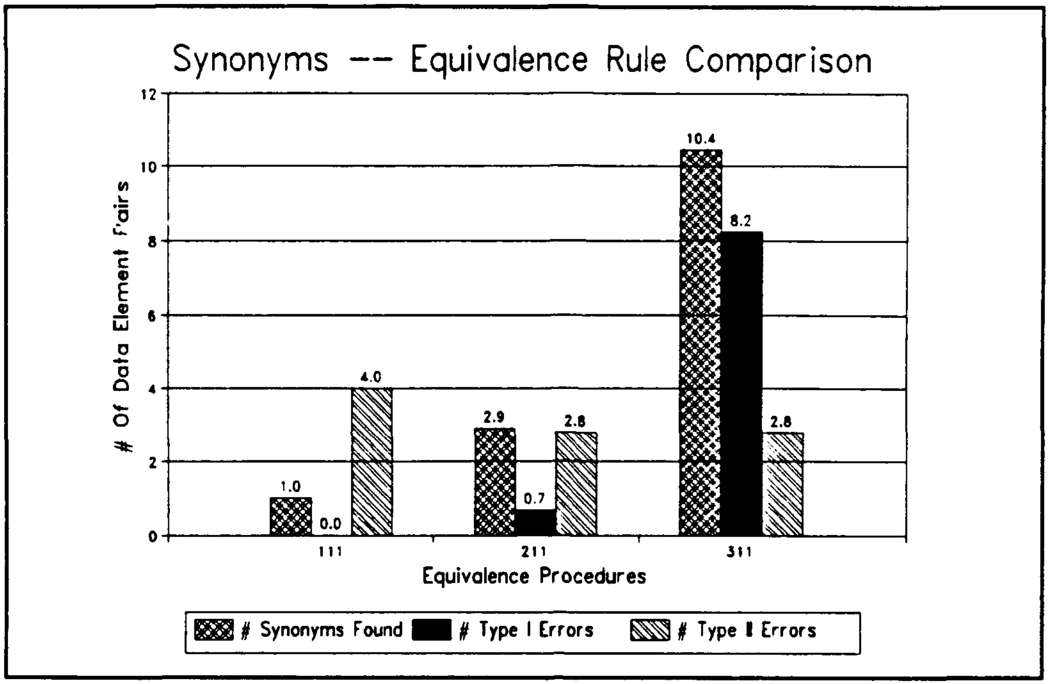


Figure 19 Synonyms -- Procedure Comparison (Component Rules 11)

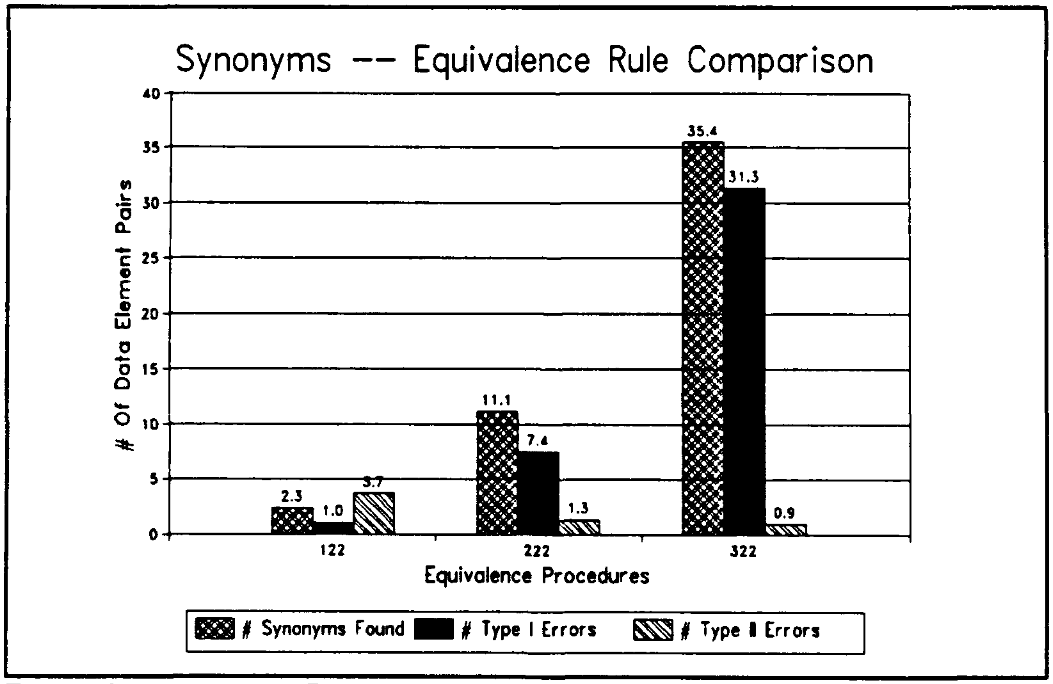


Figure 20 Synonyms -- Procedure Comparison (Component Rules 22)

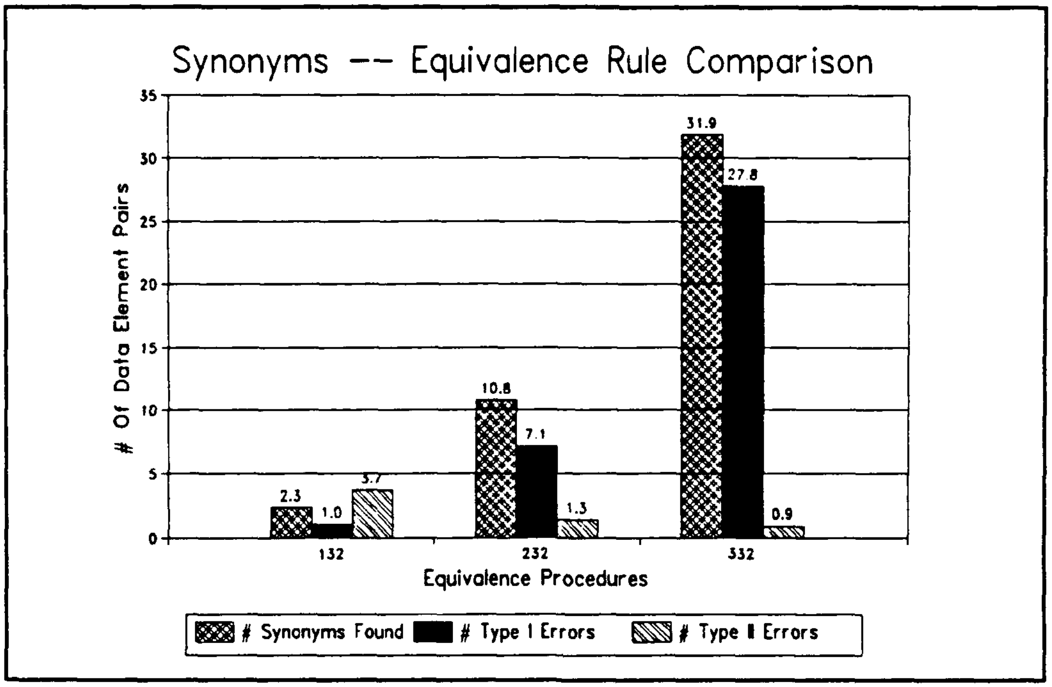


Figure 21 Synonyms -- Procedure Comparison (Component Rules 32)

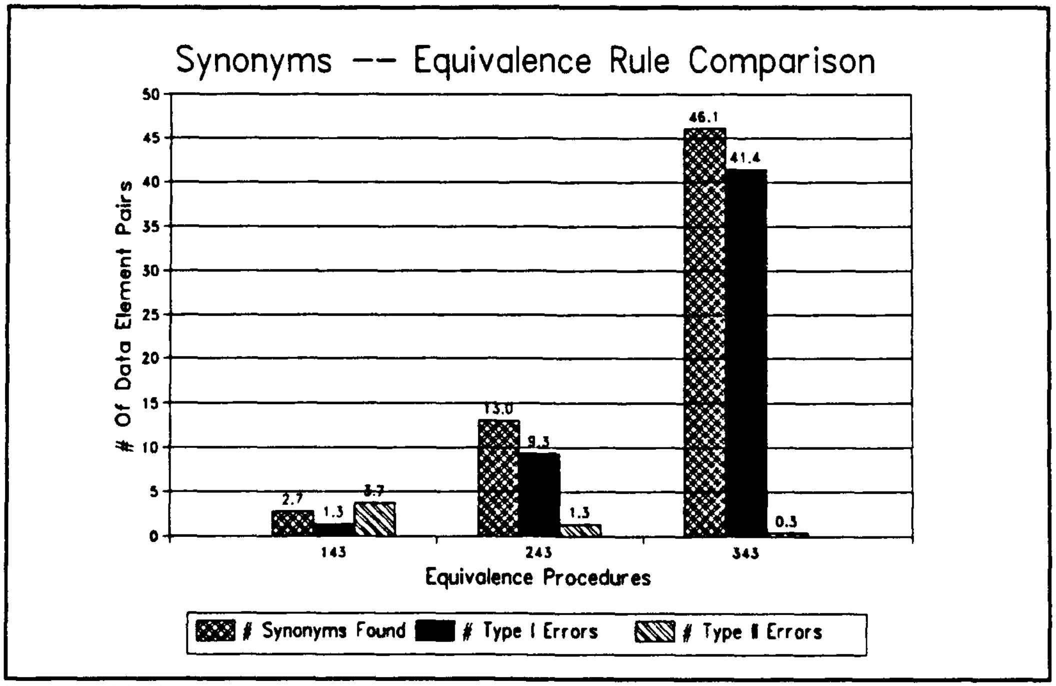


Figure 22 Synonyms -- Procedure Comparison (Component Rules 43)

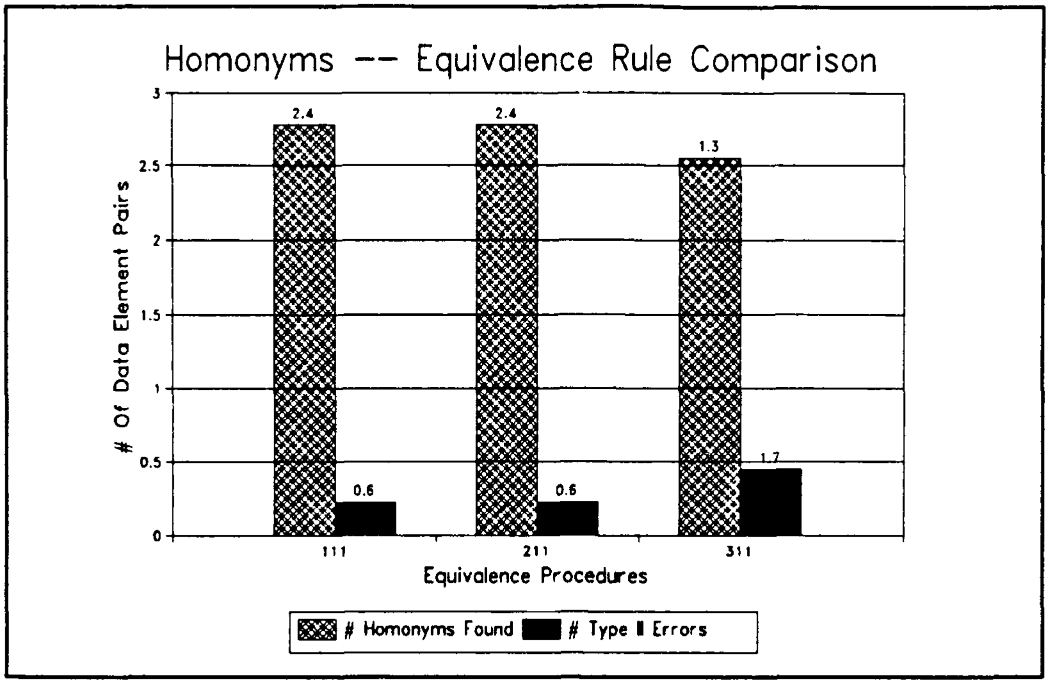


Figure 23 Homonyms -- Procedure Comparison (Component Rules 11)

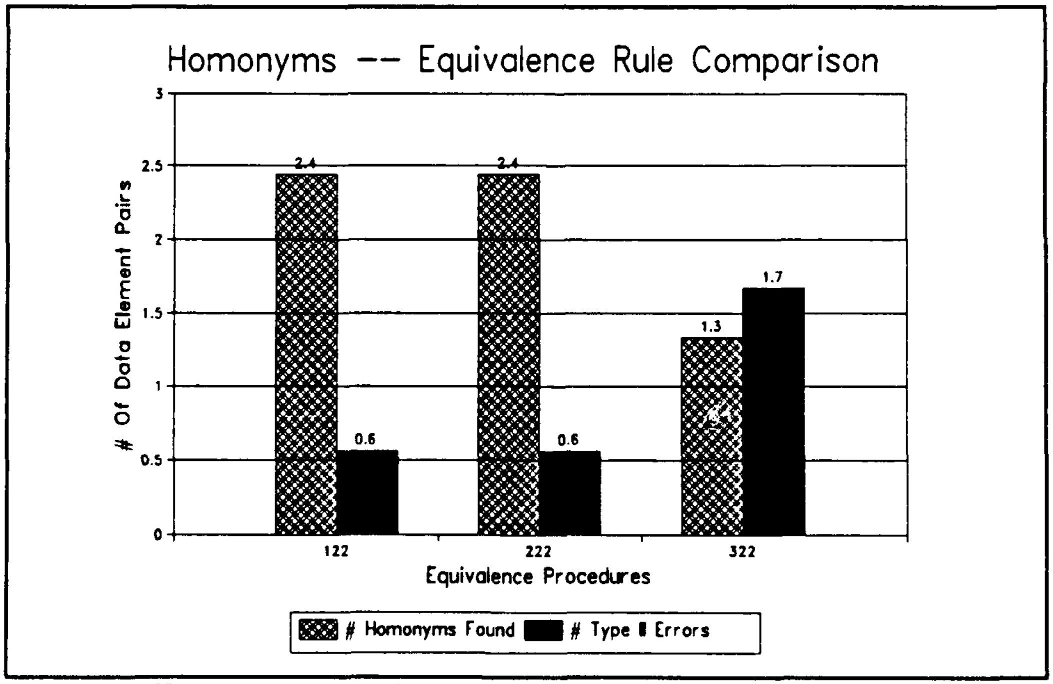


Figure 24 Homonyms -- Procedure Comparison (Component Rules 22)

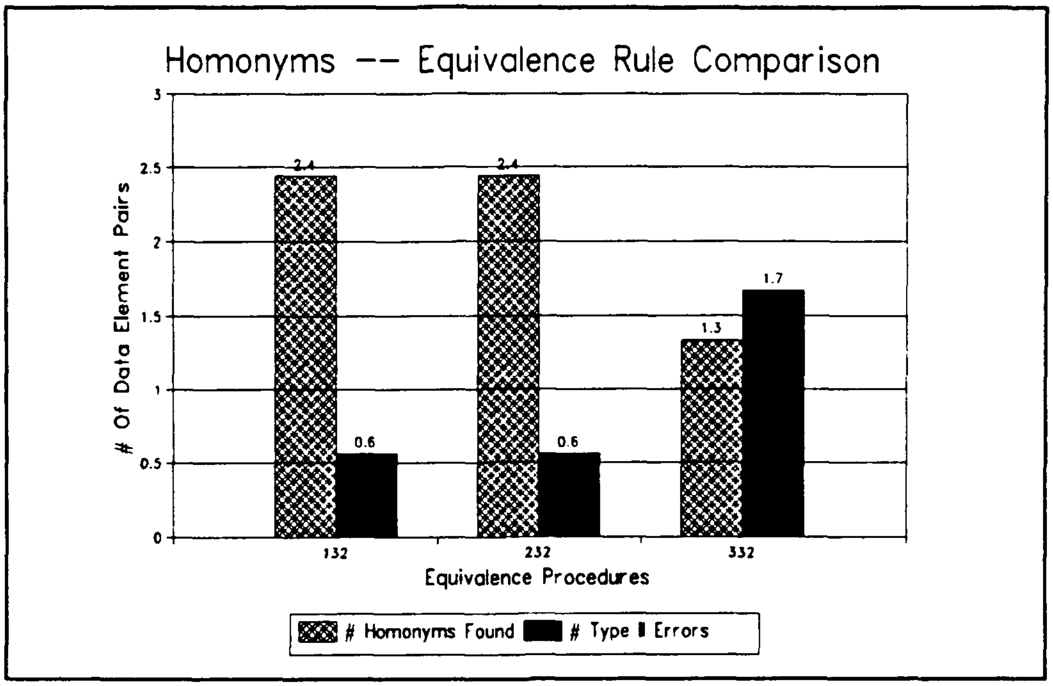


Figure 25 Homonyms -- Procedure Comparison (Component Rules 32)

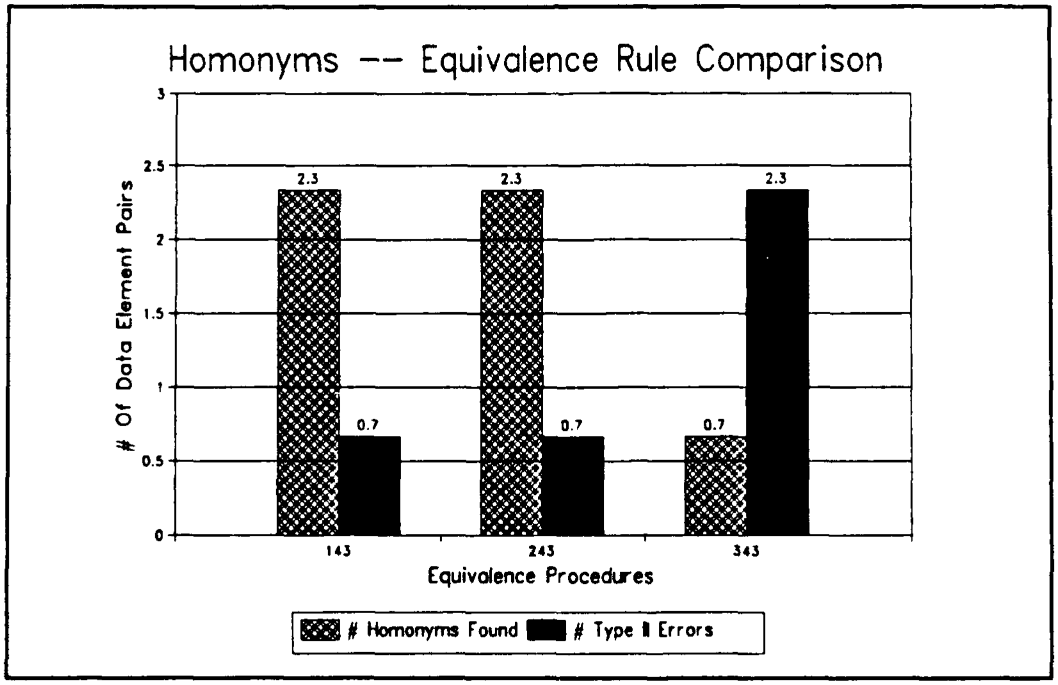


Figure 26 Homonyms -- Procedure Comparison (Component Rules 43)

NAC / CRAFT DATABASE COMPARISONS BY PROCEDURE

Synonyms
Test-111

DB-Pair	# Found	Type I	Type II
A1	2	0	3
A2	1	0	4
A3	2	0	3
B1	1	0	4
B2	1	0	4
B3	1	0	4
C1	0	0	5
C2	1	0	4
C3	0	0	5
Sum	9	0	36
Average	1.0	0.0	4.0

Synonyms
Test-211

DB-Pair	# Found	Type I	Type II
A1	5	1	1
A2	1	0	4
A3	5	1	1
B1	3	1	3
B2	1	0	4
B3	3	1	3
C1	3	1	3
C2	3	1	3
C3	2	0	3
Sum	26	6	25
Average	2.9	0.7	2.8

Synonyms
Test-122

DB-Pair	# Found	Type I	Type II
A1	5	3	3
A2	2	1	4
A3	4	2	3
B1	2	1	4
B2	1	0	4
B3	2	1	4
C1	1	0	4
C2	2	1	4
C3	2	0	3
Sum	21	9	33
Average	2.3	1.0	3.7

Synonyms
Test-222

DB-Pair	# Found	Type I	Type II
A1	18	13	0
A2	14	9	0
A3	18	13	0
B1	7	5	3
B2	8	4	1
B3	7	5	3
C1	8	5	2
C2	8	5	2
C3	12	8	1
Sum	100	67	12
Average	11.1	7.4	1.3

NAC / CRAFT DATABASE COMPARISONS BY PROCEDURE

Synonyms
Test-132

DB-Pair	# Found	Type I	Type II
A1	5	3	3
A2	2	1	4
A3	4	2	3
B1	2	1	4
B2	1	0	4
B3	2	1	4
C1	1	0	4
C2	2	1	4
C3	2	0	3
Sum	21	9	33
Average	2.3	1.0	3.7

Synonyms
Test-232

DB-Pair	# Found	Type I	Type II
A1	18	13	0
A2	13	8	0
A3	18	13	0
B1	7	5	3
B2	7	3	1
B3	7	5	3
C1	8	5	2
C2	8	5	2
C3	11	7	1
Sum	97	64	12
Average	10.8	7.1	1.3

Synonyms
Test-143

DB-Pair	# Found	Type I	Type II
A1	5	3	3
A2	2	1	4
A3	4	2	3
B1	2	1	4
B2	1	0	4
B3	2	1	4
C1	2	1	4
C2	2	1	4
C3	4	2	3
Sum	24	12	33
Average	2.7	1.3	3.7

Synonyms
Test-243

DB-Pair	# Found	Type I	Type II
A1	18	13	0
A2	17	12	0
A3	18	13	0
B1	7	5	3
B2	12	8	1
B3	7	5	3
C1	10	7	2
C2	8	5	2
C3	20	16	1
Sum	117	84	12
Average	13.0	9.3	1.3

NAC / CRAFT DATABASE COMPARISONS BY PROCEDURE

Synonyms
Test-311

DB-Pair	# Found	Type I	Type II
A1	17	13	1
A2	3	2	4
A3	20	16	1
B1	16	14	3
B2	3	2	4
B3	13	11	3
C1	8	6	3
C2	10	8	3
C3	4	2	3
Sum	94	74	25
Average	10.4	8.2	2.8

Synonyms
Test-332

DB-Pair	# Found	Type I	Type II
A1	49	44	0
A2	30	25	0
A3	53	48	0
B1	30	26	1
B2	22	18	1
B3	29	25	1
C1	22	19	2
C2	20	17	2
C3	32	28	1
Sum	287	250	8
Average	31.9	27.8	0.9

Synonyms
Test-322

DB-Pair	# Found	Type I	Type II
A1	49	44	0
A2	36	31	0
A3	53	48	0
B1	30	26	1
B2	27	23	1
B3	29	25	1
C1	29	26	2
C2	28	25	2
C3	38	34	1
Sum	319	282	8
Average	35.4	31.3	0.9

Synonyms
Test-343

DB-Pair	# Found	Type I	Type II
A1	57	52	0
A2	51	46	0
A3	53	48	0
B1	32	28	1
B2	38	34	1
B3	36	32	1
C1	46	41	0
C2	44	39	0
C3	58	53	0
Sum	415	373	3
Average	46.1	41.4	0.3

NAC / CRAFT DATABASE COMPARISONS BY PROCEDURE

Homonyms
Test-111

DB-Pair	# Found	Type I	Type II
A1	3	0	0
A2	3	0	0
A3	3	0	0
B1	3	0	0
B2	3	0	0
B3	2	0	1
C1	2	0	1
C2	3	0	0
C3	3	0	0
Sum	25	0	2
Average	2.8	0.0	0.2

Homonyms
Test-211

DB-Pair	# Found	Type I	Type II
A1	3	0	0
A2	3	0	0
A3	3	0	0
B1	3	0	0
B2	3	0	0
B3	2	0	1
C1	2	0	1
C2	3	0	0
C3	3	0	0
Sum	25	0	2
Average	2.8	0.0	0.2

Homonyms
Test-122

DB-Pair	# Found	Type I	Type II
A1	1	0	2
A2	3	0	0
A3	3	0	0
B1	3	0	0
B2	3	0	0
B3	1	0	2
C1	2	0	1
C2	3	0	0
C3	3	0	0
Sum	22	0	5
Average	2.4	0.0	0.6

Homonyms
Test-222

DB-Pair	# Found	Type I	Type II
A1	1	0	2
A2	3	0	0
A3	3	0	0
B1	3	0	0
B2	3	0	0
B3	1	0	2
C1	2	0	1
C2	3	0	0
C3	3	0	0
Sum	22	0	5
Average	2.4	0.0	0.6

NAC / CRAFT DATABASE COMPARISONS BY PROCEDURE

Homonyms
Test-132

DB-Pair	# Found	Type I	Type II
A1	1	0	2
A2	3	0	0
A3	3	0	0
B1	3	0	0
B2	3	0	0
B3	1	0	2
C1	2	0	1
C2	3	0	0
C3	3	0	0
Sum	22	0	5
Average	2.4	0.0	0.6

Homonyms
Test-232

DB-Pair	# Found	Type I	Type II
A1	1	0	2
A2	3	0	0
A3	3	0	0
B1	3	0	0
B2	3	0	0
B3	1	0	2
C1	2	0	1
C2	3	0	0
C3	3	0	0
Sum	22	0	5
Average	2.4	0.0	0.6

Homonyms
Test-143

DB-Pair	# Found	Type I	Type II
A1	1	0	2
A2	3	0	0
A3	3	0	0
B1	3	0	0
B2	3	0	0
B3	1	0	2
C1	1	0	2
C2	3	0	0
C3	3	0	0
Sum	21	0	6
Average	2.3	0.0	0.7

Homonyms
Test-243

DB-Pair	# Found	Type I	Type II
A1	1	0	2
A2	3	0	0
A3	3	0	0
B1	3	0	0
B2	3	0	0
B3	1	0	2
C1	1	0	2
C2	3	0	0
C3	3	0	0
Sum	21	0	6
Average	2.3	0.0	0.7

NAC / CRAFT DATABASE COMPARISONS BY PROCEDURE

Homonyms
Test-311

DB-Pair	# Found	Type I	Type II
A1	3	0	0
A2	3	0	0
A3	3	0	0
B1	2	0	1
B2	3	0	0
B3	2	0	1
C1	2	0	1
C2	2	0	1
C3	3	0	0
Sum	23	0	4
Average	2.6	0.0	0.4

Homonyms
Test-332

DB-Pair	# Found	Type I	Type II
A1	1	0	2
A2	3	0	0
A3	1	0	2
B1	1	0	2
B2	2	0	1
B3	1	0	2
C1	1	0	2
C2	1	0	2
C3	1	0	2
Sum	12	0	15
Average	1.3	0.0	1.7

Homonyms
Test-322

DB-Pair	# Found	Type I	Type II
A1	1	0	2
A2	3	0	0
A3	1	0	2
B1	1	0	2
B2	2	0	1
B3	1	0	2
C1	1	0	2
C2	1	0	2
C3	1	0	2
Sum	12	0	15
Average	1.3	0.0	1.7

Homonyms
Test-343

DB-Pair	# Found	Type I	Type II
A1	0	0	3
A2	1	0	2
A3	1	0	2
B1	1	0	2
B2	1	0	2
B3	0	0	3
C1	0	0	3
C2	1	0	2
C3	1	0	2
Sum	6	0	21
Average	0.7	0.0	2.3

WITHIN DATABASE COMPARISONS

Within Database Comparison -- NAC

Test	No. of Homonyms	No. of Synonyms	No. of Matches
111	35	0	10
211	30	0	15
311	30	15	15
122	26	4	19
222	16	8	29
322	10	69	35
132	26	4	19
232	18	8	27
332	12	60	32
143	22	4	23
243	12	10	33
343	6	110	39

Within Database Comparison -- NAC

Test	No. of Homonyms	No. of Synonyms	No. of Matches
111	35	0	10
122	26	4	19
132	26	4	19
143	22	4	23
211	30	0	15
222	16	8	29
232	18	8	27
243	12	10	33
311	30	15	15
322	10	69	35
332	12	60	32
343	6	110	39

Within Database Comparison -- CRAFT

Test	No. of Homonyms	No. of Synonyms	No. of Matches
111	38	0	7
211	33	6	12
311	31	28	14
122	27	5	18
222	16	43	29
322	12	111	33
132	27	5	18
232	17	41	28
332	13	99	32
143	25	8	20
243	17	46	28
343	8	128	37

Within Database Comparison -- CRAFT

Test	No. of Homonyms	No. of Synonyms	No. of Matches
111	38	0	7
122	27	5	18
132	27	5	18
143	25	8	20
211	33	6	12
222	16	43	29
232	17	41	28
243	17	46	28
311	31	28	14
322	12	111	33
332	13	99	32
343	8	128	37

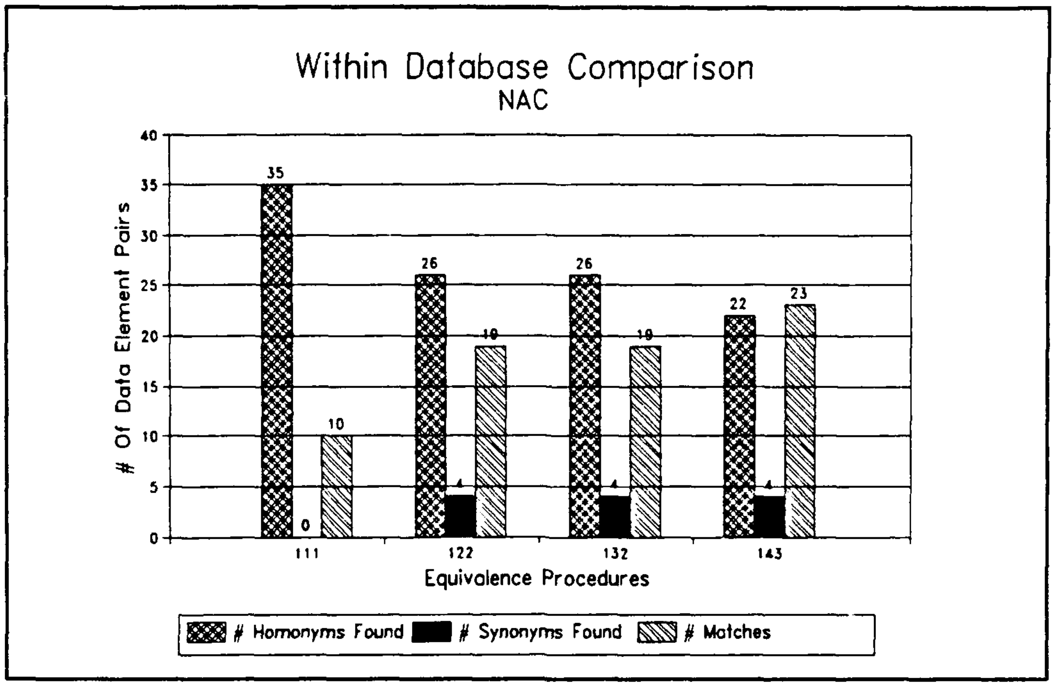


Figure 27 NAC Quiddity Sameness -- Term Equivalence Rule 1

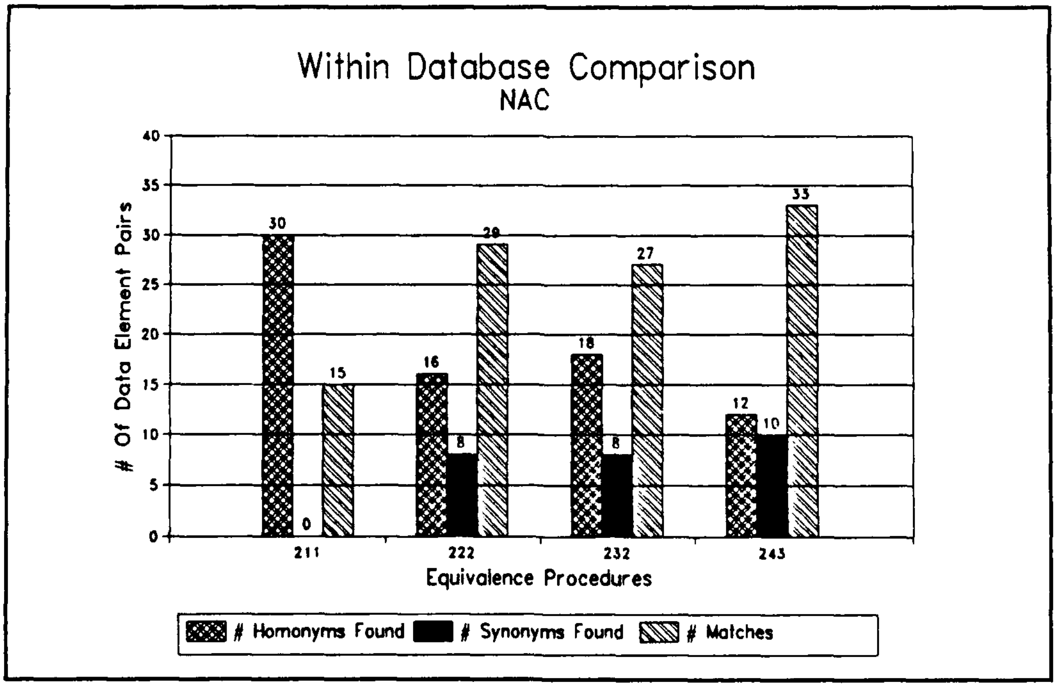


Figure 28 NAC Quiddity Sameness -- Term Equivalence Rule 2

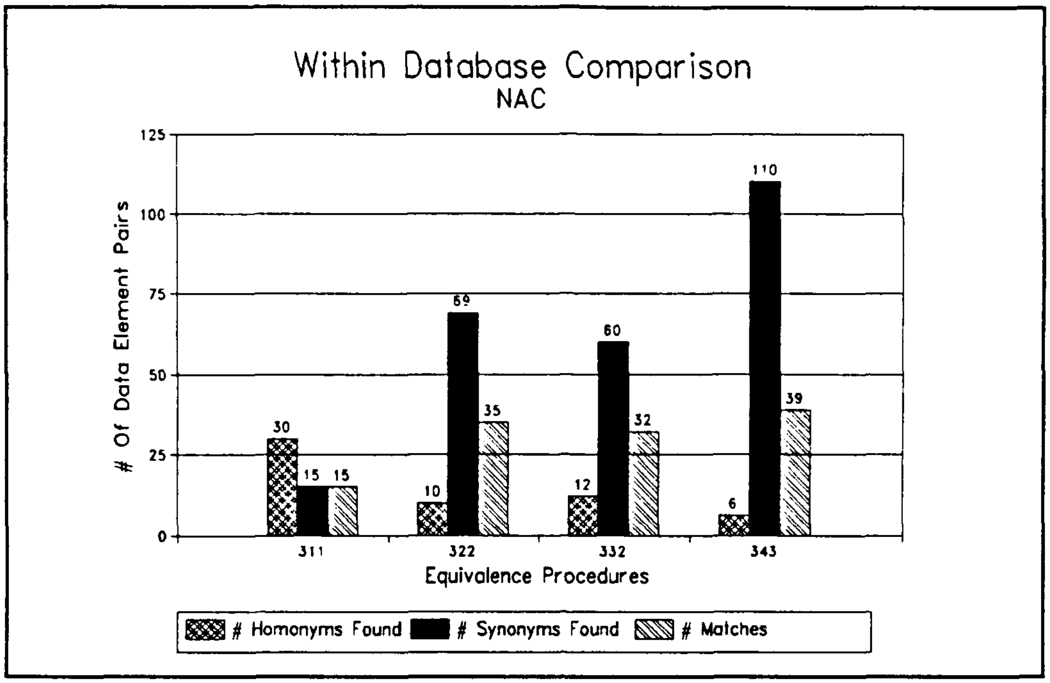


Figure 29 NAC Quiddity Sameness -- Term Equivalence Rule 3

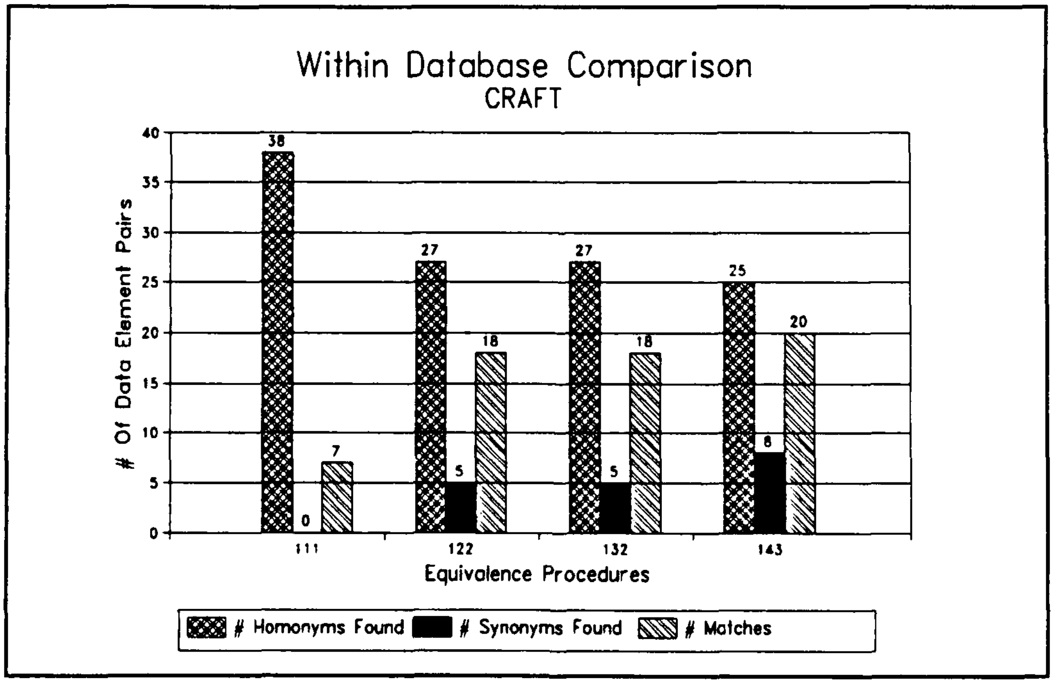


Figure 30 CRAFT Quiddity Sameness -- Term Equivalence Rule 1

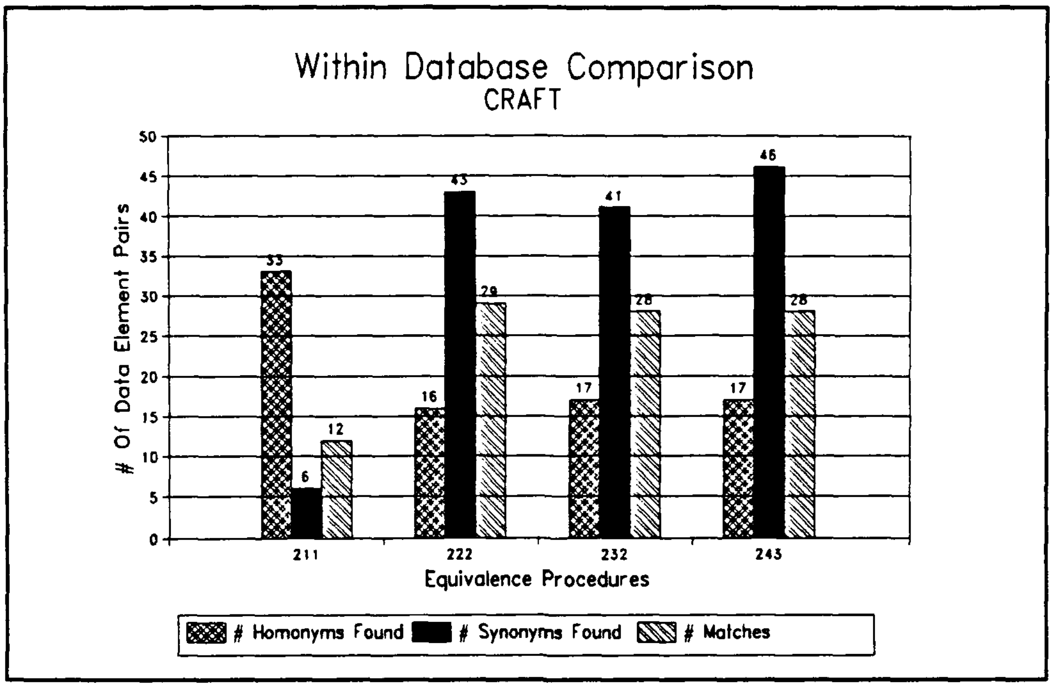


Figure 31 CRAFT Quiddity Sameness -- Term Equivalence Rule 2

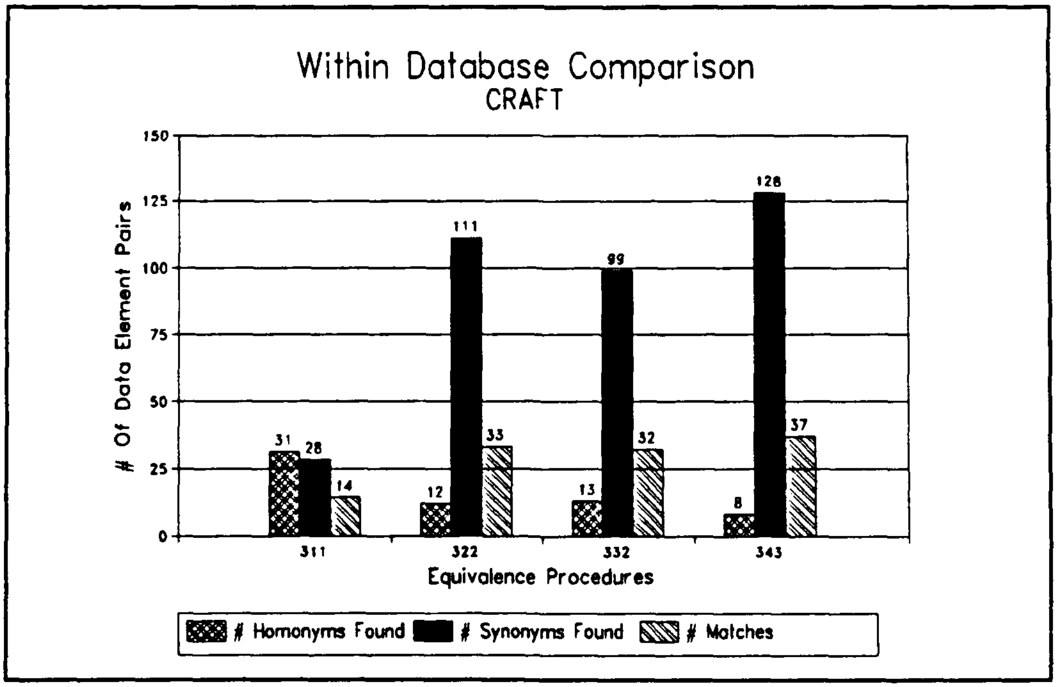


Figure 32 CRAFT Quiddity Sameness -- Term Equivalence Rule 3

LIST OF REFERENCES

Barnett, Lincoln, *The Treasure of Our Tongue*, 1st ed., pp. 29, Alfred A. Knopf, Inc., 1964.

Batini C., Lenzerini, M., and Moscarini, M., "Views Integration," in Ceri, S., *Methodology and Tools for Database Design*, North Holland, 1983.

Batini, C., Lenzerini, M., and Navathe, S., "A Comparative Analysis of Methodologies for Database Schema Integration," *ACM Computing Survey,* Volume 18, Number 4, pp. 323-364, December 1986.

Bhargava, Hemant K., Kimbrough, Steven O., and Krishnan, Ramayya, "Unique Names Violations: A Problem for Model Integration or You Say Tomato, I Say Tomahto," University of Pennsylvania, Department of Decision Sciences, Working Paper, 1990, forthcoming, *ORSA Journal on Computing*, Spring 1991.

Bouzeghoub, M., Gardarin G., and Metais, E., "Database Design Tools: An Expert System Approach," *Proceedings of Very Large Databases*, pp. 82-94, 1985.

Cardenas, A. F., *Data Base Management Systems*, 2nd e., Allyn and Bacon, Inc., 1985.

Casonova, M., and Vidal M., "Towards a Sound View Integration Methodology," *Proceedings of the 2nd ACM SIGACT/SIGMOD Conference on the Principles of Database Systems*, pp. 36-47, 1983.

Choobineh, J., Mannino M. V., Nunamaker F. F., and Konsynski, B. R., "An Expert Database Design System Based on Analysis of Forms," *IEEE Transactions on Software Engineering*, Volume 14, Number 2, February 1988.

Computer and Information Sciences Department, University of Florida, Technical Report TR-84-1, *A Methodology for Global Schema Design*, by M. V. Mannino and W. Effelsberg, September 1984.

Date, C. J., *An Introduction to Database Systems*, 5th ed., v. 1, pp. 1-102, Addison-Wesley Publishing Company, Inc., 1990.

Dayal, U., and Hwang, H., "View Definition and Generalization for Database Integration in a Multidatabase System," *IEEE Transactions on Software Engineering*, Volume SE-10, Number 6, pp. 628-645, November 1984.

Deen, S., Amin, R., and Taylor, M., "Data Integration in Distributed Databases," *IEEE Transactions on Software Engineering*, Volume SE-13, Number 7, pp. 860-864, July 1987.

DeMichiel, L. G., "Performing Operations over Mismatched Domains," *Proceedings of the Fifth International Conference on Data Engineering*, pp. 36-45, February 1989.

Dogac, A., Yuruten B., and Spaccapietra, S., "A Generalized Expert System for Database Design," *IEEE Transactions on Software Engineering*, Volume 15, Number 4, pp. 479-491, April 1989.

Elmasri, R., and Navathe, S., "Object Integration in Logical Database Design," *Proceedings of the First International Conference on Data Engineering*, pp. 423-433, April 1984.

Elmasri, R., and Navathe, S. B., "Object Integration in Database Design," *Proceedings IEEE COMPDEC Conference*, March 1984.

Elmasri, R., and Wiederhold, G., "Data Model Integration Using the Structural Model," *Proceedings of the 1979 ACM-SIGMOD Conference on Management of Data*, pp. 191-202, May 1979.

Hayne, Stephen, and Sudha, Ram, "Multi-User View Integration System (MUVIS): An Expert System for View Integration," *Proceedings of the Sixth International Conference on Data Engineering*, pp. 402-409, February 1990.

Honeywell Systems Development Division, Minneapolis, MN, Tech. Rep. CSC-86-9:8212, *Schema Integration Algorithms for Federated Databases and Logical Database Design*, by R. Elmasri, J. Larson, and S. B. Navathe, April 1986.

Kamel, Magdi N., and Hsiao, David K., "Interoperability and Integration Issues in Heterogeneous Database Environments," Naval Postgraduate School, Computer and Administrative Sciences Departments, Working Paper, 1990, forthcoming, *Information Systems Research*, 1991.

Larson, James A., Navathe, Shamkant B., and Elmasri, Ramiz, "A Theory of Attribute Equivalence in Databases with Application to Schema Integration," *IEEE Transactions on Software Engineering*, Volume 15, Number 4, pp. 449-463, April 1989.

Motro, A., and Buneman, P., "Constructing Superviews," *Proceedings of 1981 ACM-SIGMOD Conference on Management of Data*, pp. 56-64, April 1981.

Navathe, S., Sashidhar, T., and Elmasri, R., "Relationship Merging in Schema Integration," *Proceedings of the Tenth International Conference on Very Large Databases*, August 1984.

Navathe, S. B., Sashidhar, T., and Elmasri, R., "Relationship Matching in Schema Integration," *Proceedings of the Tenth International Conference of Very Large Databases*, 1984.

Navathe, S. B., Elmasri, R., and Larsen, J., "Integrating User Views in Database Design," *IEEE Computer*, pp. 50-62, January 1986.

Osborn, Patricia, *How Grammar Works*, pp. 9-67, John Wiley and Sons, Inc., 1989.

Sheth, A., Larson, J., Cornellio, A., and Navathe, S., "A Tool for Integrating Conceptual Schemas and User Views," *Proceedings of the Fourth International Conference on Data Engineering*, February 1987.

Wang, Y. R., and Madnick, S. E., "The Inter-Database Instance Identification Problem in Integrating Autonomous Systems," *Proceedings of the Fifth International Conference on Data Engineering*, pp. 46-55, February 1989.

Yao, S. B., Waddle, V., and Housel, B., "View Modeling and Integration Using the Functional Data Model," *IEEE Transactions on Software Engineering*, SE-8, Number 6, pp. 544-553, 1986.

Zviran, Moshe, and Kamel, Magdi N., "A Methodology for Integrating Heterogeneous Databases in a Hospital Environment," Naval Postgraduate School, Department of Administrative Sciences, Working Paper, No. 89-10, 1989.

BIBLIOGRAPHY

Chase, Stuart, *Power of Words*, Harcourt, Brace, and World, Inc., 1954.

Chomsky, Noam, *Aspects of the Theory of Syntax*, The M.I.T. Press, 1965.

Cooper, William S., *Set Theory and Syntactic Description*, Mouton and Co., 1964.

Hayakawa, S. I., *Language in Thought and Action*, 2nd e., Harcourt, Brace, and World, Inc., 1964.

Manser, Martin, *The Guinness Book of Words*, Guinness Publishing Ltd., 1988.

Pei, Mario, *The Story of the English Language*, Revised Edition, J. B. Lippincott Company, 1967.