CABSYDD: Case-Based System for Database Design

JOOBIN CHOOBINEH AND AMBER W. LO

JOOBIN CHOOBINEH received his Ph.D. in Management Information Systems from the University of Arizona in 1985. He is an Associate Professor in the Department of Information and Operations Management at Texas A&M University. Dr. Choobineh’s research areas include information systems security, data modeling, electronic commerce, integration of data and mathematical models, and creation of intelligent systems to solve organizational problems. His work has appeared in journals such as Communications of the ACM, Journal of Management Information Systems, IEEE Transactions on Software Engineering, INFORMS Journal on Computing, Decision Support Systems, Information Systems, Information and Management, Annals of Operations Research, Omega: The International Journal of Management Science, Database for Advances in Information Systems, International Journal of Operations & Production Management, Information Systems Management, Journal of Database Management, Information Strategy, and Database Engineering. Since 1986, Dr. Choobineh has successfully obtained in excess of $1.2 million in research and educational grants. He has served as the chair of eight and committee member of eleven Ph.D. students. Dr. Choobineh is currently an Associate Editor of INFORMS Journal on Computing.

AMBER LO received her Ph.D. in Business Analysis and Research from Texas A&M University in 1994. She was Assistant Professor in Business Administration at Millersville University of Pennsylvania from 1993 to 1999 (tenured 1998). She is currently a Research Scientist at Knowledge Based Systems, Inc. Dr. Lo’s research interests include artificial intelligence, knowledge management, data modeling, and case-based reasoning. In addition to publishing in academic outlets such as Annals of Operations Research, Journal of Database Management, and conference proceedings, she has published over 20 articles on interpersonal relationship issues. Dr. Lo has recently obtained a Small Business Innovative Research (SBIR) grant from the U.S. Army.

ABSTRACT: The majority of the reported research and development efforts on automated techniques and tools for conceptual database design have focused on design from first principles. Very few have used case-based reasoning, where cases of conceptual design are stored, indexed, and used for future designs. Furthermore, there is a general lack of reported research on validating and verifying such systems. In this paper, we describe our approach in using case-based reasoning for conceptual database design. To test and demonstrate the feasibility of our approach and its theoretical foundation, two prototype systems were constructed. In the absence of existing matching conceptual design constructs, the first system uses first principles of conceptual design to assist a human designer in arriving at a design for a new problem. In contrast, the second system uses constructs from previously stored design cases. The two are tightly integrated. A novel approach in structuring the case base was developed. Unique
aspects of the case-base architecture and its learning mechanism are described. In order to measure user preference, an experiment was designed and conducted. Findings indicate that reuse of schemata not only is preferred by the users over the design from the first principles, but also results in fewer errors.

KEY WORDS AND PHRASES: automated database design, case-based reasoning, conceptual database design, database design.

CAPTURING REQUIREMENTS FOR DEVELOPING an information system has been a subject of intense research since the early days of using computers for business applications (see, e.g., [1, 2, 7, 48]). Developing a database system to serve the information needs of its users is an important component of overall system design. It includes (1) requirements collection and analysis, (2) conceptual database design, (3) logical database design, (4) physical database design, and (5) implementation [20]. The first four steps make up the database design process through which informal and imprecisely defined end-user requirements are turned into a rigidly defined database structure, a set of operations, and a set of system software/hardware configurations. This process is knowledge intensive, complex, and difficult (see, e.g., [48]). It requires the use of heuristics, judgment, and experience. Once the system is installed, the design cannot be easily modified. Because of these characteristics, many intelligent computer-aided database design tools have been created. Table 1 shows a list of those reported in the literature. In Lo and Choobineh [30], we provided a detailed analysis of many of these and other systems for database design.

Two kinds of technologies are used in constructing these kinds of database systems: expert systems and case-based reasoning. Expert systems use first principles of knowledge to solve a problem. Each new problem is solved by applying the rules and heuristics of conceptual design. No previous similar cases are used to solve a new problem. In case-based reasoning, past problems and their solutions are used to solve a new problem. Conceivably, this approach will be more effective and will result in higher productivity, because a new problem can be solved much faster when a similar previously solved problem is used as a starting point. As reported in Table 1, except for common-sense business reasoning (CSBR) [46] and design expert system for database schema (DES-ES) [36], which use case-based reasoning, all other systems use first principles of knowledge. Later, we will compare these two to our research.

The central hypothesis of this research was that an intelligent case-based reasoning system for conceptual database design would be more effective and would result in higher productivity than a comparable system based on first principles of knowledge. Similar to most research prototypes, current commercial systems implement first principles knowledge for database design. They assist a designer, but do not contain a store of past schemata. On the other hand, a properly designed case-based database design tool will (1) store and retrieve design solution cases, (2) help a designer in
<table>
<thead>
<tr>
<th>System</th>
<th>Reference citation</th>
<th>Requirements</th>
<th>Conceptual</th>
<th>Logical</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODASYS</td>
<td>Antony and Batra [1]</td>
<td>—</td>
<td>*</td>
<td>—</td>
</tr>
<tr>
<td>SECSI</td>
<td>Bouzeghoub [3]</td>
<td>—</td>
<td></td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>Bouzeghoub and Gardarin [4]</td>
<td>—</td>
<td></td>
<td>—</td>
</tr>
<tr>
<td>GAMBIT</td>
<td>Braegger et al. [5]</td>
<td>—</td>
<td>*</td>
<td>—</td>
</tr>
<tr>
<td>OICSI</td>
<td>Cauvet et al. [9]</td>
<td>*</td>
<td></td>
<td>—</td>
</tr>
<tr>
<td>EDDS</td>
<td>Choobineh et al. [13]</td>
<td>—</td>
<td>*</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Choobineh et al. [14]</td>
<td>—</td>
<td></td>
<td>—</td>
</tr>
<tr>
<td>FOBFUDD</td>
<td>Choobineh and Venkatraman [12]</td>
<td>—</td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>GESDD</td>
<td>Dogac et al. [19]</td>
<td>*</td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>Knowledge-based</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>information support</td>
<td>Falkenberg et al. [21]</td>
<td>—</td>
<td>*</td>
<td>—</td>
</tr>
<tr>
<td>PROEX</td>
<td>Obretenov et al. [34]</td>
<td>—</td>
<td>*</td>
<td>—</td>
</tr>
<tr>
<td>DES-DS</td>
<td>Paek et al. [36]</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VCS</td>
<td>Storey and Goldstein [44]</td>
<td>*</td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>CSBR</td>
<td>Storey et al. [46]</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MODELLER</td>
<td>Tauzovich [47]</td>
<td>—</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>Form definition system</td>
<td>Choobineh et al. [13]</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tseng and Mannino [50]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHRIS</td>
<td>Tucherman et al. [51]</td>
<td>—</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>EXIS</td>
<td>Yasaki and Zdardko [56]</td>
<td>—</td>
<td></td>
<td>*</td>
</tr>
</tbody>
</table>

Notes: * The design step is covered by the tool; — the design step is not covered by the tool.
finding a stored case that is similar to a new problem, and (3) assist a designer in adapting the solution to the similar stored case to arrive at a solution for the new problem.

To test this hypothesis, three specific objectives were established. The first and second objectives were the construction of an expert system and a case-based reasoning database design tool, respectively. The third objective was to conduct an empirical test to verify the central hypothesis. A supplementary, but very important, objective was that the two systems must be tightly integrated such that the user would not know which reasoning engine was being used. Such transparency requires the same user interface. Furthermore, the two should be equal in all their characteristics except for their reasoning engines.

There were two fundamental reasons for establishing the supplementary objective. One is that to bootstrap the case-based system, the use of the first principles knowledge (which is contained in the expert system) is required; that is, if there are no existing similar cases in the case base, the system should be able to assist in the creation of that very first case through the use of the expert system. The other reason is that, even in a situation when a large part of a new problem is designed by case-based reasoning, there still remain some parts that require reasoning from the first principles.

We constructed SYDD (system for database design) to meet the first objective. SYDD implements first principles knowledge of database design. We finished construction of CABSYDD (case-based system for database design) to meet the second objective. CABSYDD is an instantiation of a case-based approach to database design. It learns from experience—It improves its inference ability each time a new problem is encountered. The third objective was achieved by conducting an experiment that assessed the perceived effectiveness as well as the productivity of CABSYDD compared to SYDD. In this paper, we focus on describing the first two objectives. Details of the third objective are presented in Choobineh and Lo [11].

Case-Based Reasoning

Humans make use of past experience to deal with new problems. Making changes to a similar solution to arrive at a solution for the new problem requires less work than starting from nothing. Performing a task or solving a problem by using past experiences consists of four steps (see, e.g., [28, 40]): (1) task description, (2) retrieval and reconstruction of a past experience, (3) application of past experience, and (4) learning. Over time, one’s store of past experiences grows and becomes richer.

A relevant theory is that of concept categorization in cognitive psychology. Concept categorization is a method for grouping specific experiences into different categories for efficient retrieval [31]. To facilitate the retrieval of the correct concept, the content of memory is organized in conceptual categories, which are based on similarities in meaning. Within each category, similar concepts are grouped together. Searching for a particular concept in memory under its own category is superior to a sequential search through every specific concept.
Schank [41] proposed the idea of an artificial system that can be modeled after the human problem-solving processes. He suggested that this kind of artificial system should be capable of maintaining a dynamic memory system to organize specific experiences as it encounters new problems and events. Furthermore, it should be adaptable, alterable, naturally extensible, and capable of self-modification. As new experiences are encountered, the structure and contents of this “dynamic memory” change over time. The two major components are stored experiences and the organization of these experiences. Experiences are organized to facilitate correct and efficient future retrieval. They are grouped under distinct categories. To solve a new problem, the system should be able to maintain, use, and provide a set of clues in identifying matching categories, as well as specific experiences within those categories.

Case-based reasoning is based on the above theories. It is an attempt to automate human problem solving [40] by organizing past experiences [31] in a dynamic memory [41]. The most basic components of a case are a problem statement and its corresponding solution. A case base consists of a set of organized cases. Cases are organized by a set of selected features. These features should be chosen to facilitate efficient matching, with a high predictive power of choosing the closest case to the problem in hand.

The organization scheme should be designed to achieve two goals. The first is to facilitate efficient base case retrieval during problem solving. The second is to properly integrate new cases into the case base. This integration constitutes learning. A new problem is solved by identifying a similar problem that was previously solved. The solved problem is adapted to arrive at a solution for the problem under study. A solved problem is learned if it is “new enough.” Learning is achieved by saving and properly indexing the parameters and solution of the new case. All such cases are stored in the case base for future retrieval. Each problem, together with its solution and index values, is called a “case.” The reasoning process can be divided into four major steps (see [29, 39]): (1) problem description, (2) base case searching and retrieval, (3) base case adaptation and new solution evaluation, and (4) learning. Figure 1 shows the overall reasoning process.

Case-based reasoning has been used in different application domains. Tasks that are suitable for case-based reasoning include classification, planning, design, and justification of arguments. Literature surveys and sample applications can be found [25, 26, 28, 29]. An example of a commercial case-based reasoning engine can be found at Haley Enterprise Inc. [24].

Semantic Data Model for Conceptual Database Design

THE ENHANCED ENTITY RELATIONSHIP (EER) MODEL of Elmasri and Navathe [20] was the basis of the data model for this research. The choice was based on the fact that EER, or slight variations of it, is widely used in conceptual data modeling literature as well as in tools of practice. Aside from these two considerations, any other varia-
A new problem

Step 1
Problem Description
The new problem with index values

Step 2
Base Case Searching & Retrieval

A useful base case

Step 3
Base Case Adaptation & New Solution Evaluation

New solution

Step 4
Learning

Legend

- Process
- Input/output
- Long-term storage
- Process flow
- Input-output flow

Figure 1. The General Process of Case-Based Reasoning

Application of the Entity Relationship model would have been equally suitable for our research. (Equally important is the fact that case-based reasoning could be applied to other data models.)

Figure 2a is the EER diagram of the conceptual schema of a hypothetical public university library. Details of this case are presented in Appendix A. Figure 2b contains the details of the diagram presented in Figure 2a.

Constructs of the EER model are as follows. Entities are represented by rectangles, relationships by diamonds, and generalization hierarchies by circles and arrows. In a
generalization hierarchy, one of the letters, "D" (disjointness of subentities) or "O" (overlapping subentities), must be placed inside the circle. Moreover, the letter "U," if present by the lines between a circle and its supertype, denotes that the supertype must be the union of the subtypes. (Note that "U-Type" in Figure 2a denotes the attribute name whose values partition USER into its subtypes. The letter U in "U-Type" should not be confused with a stand-alone "U" that denotes the union constraint. Neither of the two generalizations in Figure 2a is constrained by the union of its subtypes.)
Through specialization, a cluster of subclass entity types is formed from a superclass entity type. A superclass may have several clusters, where each is defined by a different attribute. For example, a library holding may have two subclass clusters. One is based on the type of holding (Book for Circulation, Reference, Periodical, or Video) and another on the locality of interest (Local and Nonlocal holdings). The superclass entity type at the highest level of the generalization hierarchy is called the "root-level" entity type.
Following Storey and Goldstein [44], to reduce computational complexities, we restricted all relationship types to be binary. Minimum and maximum cardinalities are placed on the relationship lines. For example, the min-max cardinality of 0:3 in Figure 2a for the entity USER in the relationship CHECKS-OUT-R denotes the constraint that a user can check out a minimum of 0 and a maximum of 3 references. Attributes are properties of entities or relationships. In Figure 2b, key attributes are underlined.

To facilitate case-based reasoning, four additional constructs were introduced. The first three further characterize entities. The last pertains to relationships. These constructs are domain, chunk, role, and activity. Domain is a short declarative phrase that further describes the scope of an entity type. It is used to provide additional information to human designers besides just an entity's name. For example, the domain of "HOLDING" is "All library holdings" and that of "REFERENCE," a subclass of "HOLDING," is "All books in Reference." The collection of a root entity type and all its clusters is called a chunk of entity types. An example is the chunk of "USER" in Figure 2a. It consists of the collection of "USER," "STUDENT," "STAFF," and "COMMUNITY USER." In this example, there is only one cluster. To illustrate the relationship between a chunk and its clusters, in Figure 3 we show a hypothetical hierarchy of super- and subentity types. The entire hierarchy is a chunk. E1 is the root-level entity. There are four clusters in this chunk. To identify them for the reader, each is placed in an oval. Cluster number 1 consists of E2, E3, and E4. Each is a subclass of E1. Similarly, cluster 2 consists of E5 and E6, where each is a subclass of E2. Similar interpretation applies to clusters 3 and 4. Role is a short generalized statement representing the characteristics of a chunk. An example of a role statement is: "Members of the library," which defines the role of the entity type "USER" of Figure 2a. Roles are used by CABSYDD to decide whether an entity type is applicable to a new case. All the entity types in the same chunk share the same role statement. In Figure 2, all users are members of the library regardless of the subclass to which they belong. The concepts of "chunk" and "role" together are analogous to the single concept of "combined class" as presented in Purao et al. [38]. Activity is a statement describing a relationship. An example is: "A user checks out a book for circulation."

Case-Based Approach to Conceptual Database Design

Di Battista et al. [18] were among the first to propose the idea of creation and maintenance of schema libraries. Case-based reasoning is the inference process to effectively create, maintain, and use such libraries. In designing automated systems for database design, Bouzeghoub [3] explained the desirability of inclusion of first principles of knowledge, as well as modular knowledge bases that can be expanded or contracted. The advantages of tools that combine first principles of knowledge of database design and case-based reasoning include: reuse of previous designs, automated reasoning in assisting a designer in creating a new design, use of best practices of design, and assurance of a correct design.
Our approach in creating CABSYYDD was based on the suggestions of Bouzeghoub [3] and Di Battista et al. [18]. The approach is similar, but not identical, to approaches used in learning from patterns of stereotypical objects. For instance, in object-oriented analysis and design, libraries of patterns can be used to design new systems. Variations of these libraries are being compiled and exist today (see [16, 22]). An example of a system that uses stereotypical patterns to design object-oriented solutions is APSARA [37]. The use of case-based reasoning is similar to the use of stereotypical patterns. Both start from an existing design. One adapts a case; the other, a pattern. The difference is that the former uses real cases that are dynamically compiled, whereas the latter uses the same stereotypical pattern for all future problems. In what follows, we will describe the implementation of this approach in CABSYYDD.

The Case Base of CABSYYDD

Figure 4 is a partial representation of the case base in CABSYYDD. The lower part of the figure, below the Department level, contains examples of departments and cases.
Here, a case is a complete, stand-alone, and previously defined conceptual schema for one particular functional department of an enterprise. Departments that perform similar functions in enterprises of the same industry share many common entities and relationships. Conceptual schemata defined at this level should be highly reusable. Moreover, departmental schemata of enterprises in related industries share many common elements.

For example, the schema for the library of a secondary school can be used as a basis for defining a schema for the library of a college. Therefore, it is logical to classify conceptual schemata for functional departments according to the industries to which they belong. We chose the North American Industry Classification System (NAICS).
[33] as the basis for case-base organization. NAICS [33] is a complete and unbiased hierarchical representation of all industries in North America.

The initial case-base architecture is the three-level hierarchy of NAICS. These are sector, subsector, and industry group. Parts of these three levels are shown as the top three layers in Figure 4. We added layers below the third for case-based reasoning and learning. Sector is at the highest level of the hierarchy of NAICS. There are 16 sector names. Three of these sectors ("agriculture, forestry, fishing, and hunting," "educational services," and "public administration") are shown in Figure 4. Each sector name is used as an index value in the case base of CABSYYDD. Under each sector name is the index subsector. For example, under the "educational services" sector, there is only one subsector with, incidentally, the identical label of "educational services." Under each subsector is the index industry group. For example, under "educational services" there are seven industry groups. Three of these are shown in Figure 4.

For this research, we added "departments" to the hierarchy. Each industry group consists of a set of departments. For example, the typical functional departments of many kinds of schools include: the library, the registration office, and the financial aid office. A user supplies the name of a department when a new conceptual schema is stored as the first case under that department. Note that "industry group" and "department" are orthogonal concepts. Later, we discuss when a neighboring industry will be used to search for similar departments.

We use the term predefined indices to refer to indices that correspond one-to-one to the top three layers in Figure 4. To facilitate identification of "good" matches for a new problem, further indexing (below the department level) is required. Unlike predefined indices, these are learned by the system. Hence, we call them "learned indices." Department names are not learned. Their names are supplied directly by users. Each learned index is a root-level entity type name of a case. Initially, when the system contains no cases, the case base consists of the codification of the hierarchical classification in NAICS [33]. Nothing exists below the "Industry Group" level (see Figure 4). New departments, cases, and learned indices are added as the case base grows with each new design problem. Cases are the leaf nodes of the case base.

The learning algorithm records the differences between the cases instead of their similarities. Given that very similar cases will be stored under the same department, favoring similarities will result in the creation of a large number of index values. Overlearning and a large search space will result. A larger-than-necessary search space would cause an inefficient search. Clearly, one should be judicious in storing similarities. Accordingly, our learning heuristic assumes that cases under the same department have many more similar entities than dissimilar ones. Hence, its algorithm favors differences between the cases over their similarities. (See the example below and "The Design Process" section.)

Note that it is not necessary to acquire a collection of cases to start using the system. If there are no existing cases, SYDD (instead of CABSYYDD) is automatically invoked to assist a designer in creating a new design. As cases are added, more problems are designed through CABSYYDD. Note that the indexing and learning method are original to this research.
Example: In Figure 4, cases are tied to learned indices by a “y” (for yes) or an “n” (for no). An index value of “y” means this case contains that root-level entity type. A value of “n” means that the entity is not part of this case. This method of indexing is original to this research. In Figure 4, at this point in time in the life of the case base, there are three cases under the “library” department, and three learned indexes: (1) Librarian, (2) Study Room, and (3) Locker. Below, we show the value of each learned index for each case.

<table>
<thead>
<tr>
<th></th>
<th>Case 1</th>
<th>Case 2</th>
<th>Case 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Librarian</td>
<td>y</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>Study Room</td>
<td>n</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>Locker</td>
<td>n</td>
<td>n</td>
<td>y</td>
</tr>
</tbody>
</table>

Referring to Figure 4 and using the library department as an example, we now show how an empty case base was populated with three cases. In subsequent sections, we provide additional details. We will use the following conventions in showing the names of various elements: entity names are represented by all uppercase letters, learned indices by upper and lowercase letters, and department level indices by all lowercase letters.

Suppose case 1 is the first case ever being designed. Through interaction with the user, it is determined that a new department, library, has to be created. Further, suppose that case 1 has three root-level entity types: HOLDING, USER, and LIBRARIAN. Because there are no cases in the knowledge base, SYDD is used to construct the first case from first principles knowledge. Partial structure of the case base, following case 1’s design, is shown in Figure 5a. There is only one case with three entities: HOLDING, USER, and LIBRARIAN.

Next, case 2 arrives under the same department. Case 1 is chosen for adaptation, because there are no other cases under the library department. Following interaction with the designer, it is determined that case 2 has two root-level entity types that are very similar to HOLDING and USER. Case 2 has a third entity, STUDY ROOM, for which there is no similar entity in case 1. HOLDING and USER of case 1 are adapted and transferred from case 1 to case 2. The differences between the two cases are the entities LIBRARIAN and STUDY ROOM. These will be recorded as learned indices (see Figure 5b). In Figure 5, the index values for each case are written on the lines connecting the cases to the indices.

Next comes case 3 under the same department. (Case 3 is presented in Appendix A.) Which case, 1 or 2, is a better match? At this point in time, there are two indices under “library” in the knowledge base: Librarian and Study Room. Interaction with the designer determines that there is no similar root-level entity type to that of Librarian, but there is one similar to Study Room. Therefore, for case 3, the values for the Librarian and Study Room indices will be “n” and “y,” respectively. Case 1 and case 2 each have one match for case 3. Because there is a tie, each has an equal chance of being selected as a “good” case for adaptation. One is chosen and adapted to generate a design for case 3.
a. The library department when there is only one case.

b. The library department after addition of case 2. Note the learning of two indexes, "Librarian" and "Study Room," and the four corresponding index values.

c. The library department after the addition of case 3. Note the learning of the additional index "Locker" and the three corresponding index values.

Figure 5. An Example of Learning in CABSYYDD
Case 3 has another root-level entity type, LOCKER. There is no similar entity to LOCKER in the other two cases. An index by this name, Locker, is created to record this difference (see Figure 5c).

The Design Process

The four steps of case-based reasoning, shown in Figure 1, were implemented in CABSYDD as follows.

*Step 1—Problem description:* Information about a new conceptual schema to be defined is elicited. CABSYDD asks for a value for each relevant predefined and learned index.

*Step 2—Base case searching and retrieval:* CABSYDD compares the characteristics of the new case to those of the existing cases and selects the one deemed most similar. Base case searching and retrieval are carried out in a best-first search manner. There are three possibilities.

*Possibility 1:* A single department matches a new case.

*Example:* In Figure 5a, only one case exists for “library.” This single case will be retrieved to be the base case for case 2.

*Possibility 2:* There is more than one case for a department. When this occurs, each case should have been further indexed by some learned index(es). The one that has the most matching learned index values would be chosen as the base case. A similarity value is calculated for each candidate case. The metric favors matching learned index values over nonmatching ones. It weights all indices equally. The final score for each case is calculated as the total of all index scores divided by the number of indices for that department. A matching index value is scored 1. A nonmatching value is scored 0. The case with the highest score is selected as the base case.

*Example:* Consider case 4, which is presented in Appendix B. As shown in Figures 4 and 5c, there are three existing cases under the industry group “colleges, universities, and professional schools” and the department “library.” Below, we show the index values for each of the cases. Also shown is the computed similarity value of each of the existing cases for the new case, case 4.

<table>
<thead>
<tr>
<th></th>
<th>Case 1</th>
<th>Case 2</th>
<th>Case 3</th>
<th>Case 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Librarian</td>
<td>y</td>
<td>n</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>Study Room</td>
<td>n</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>Locker:</td>
<td>n</td>
<td>n</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>Similarity value</td>
<td>0/3</td>
<td>2/3</td>
<td>3/3</td>
<td>—</td>
</tr>
</tbody>
</table>

Case 3 is chosen as the base case for adaptation because it has the highest similarity value to the new case, case 4. Note that the algorithm uses only similarities that are learned.
Possibility 3: No cases exist for a department. An attempt will be made to find a case for the same functional department among other industry groups of the same subsector. If the neighboring industry groups collectively have one or more cases for that department, then the solution is identical to those that were presented under the first and second possibilities above. If no industry group has a case for that department, then SYDD is invoked to define a new schema from first principles knowledge of database design.

Example: A new case is to be designed for the library of a college. The case base does not have any existing library schema for the "colleges, universities, and professional schools" industry group. A search is launched within the neighboring industry groups: "elementary and secondary schools," "junior colleges," and so on. If a case is found, then it will be used. If not, then SYDD is invoked.

Step 3—Base case adaptation and new solution evaluation: There are two general methods of adaptation: structural adaptation (or substitution) and derivational adaptation (or transformation) [39, p. 41]. In structural adaptation, based on the roles that they play, corresponding elements of the old solution and the new problem are identified. The old solution is then adapted to arrive at a solution for the new problem. In derivational adaptation, a new solution for the new case is derived by applying the reasoning path that had been taken to reach a solution for the old problem. We chose structural adaptation instead of derivational adaptation because defining a conceptual database schema is more about deriving the structure of a schema than about specific paths of reasoning. Further, the proper database conceptual design reasoning steps are quite uniform; the entity type chunks are first defined, then the attributes of each entity type, followed by the relationship types and their attributes.

There are two strategies for structural adaptation: direct substitution or parameter adjustment. In direct substitution, the corresponding elements of the existing case replace those of the new problem. Then, through an iterative process of evaluate-and-repair, a final solution is reached. In the parameter adjustment strategy, the differences between the old and new problem are examined. Appropriate adjustments are made to the parameters of the old solution to arrive at a new solution [29]. For this research, we chose the parameter adjustment strategy, because it is more efficient and natural to analyze an old feature and adjust it to become the correct new feature than to blindly transfer old features to a new schema and go back to do repair work. In our case, the parameters are the properties of a similar existing case. These include entities, relationships, attributes, cardinalities, chunks, roles, and other detail properties of the model. Adjustments include redefining these parameters for the new case.

Recall that in the Base-Case Searching and Retrieval step, only root-level entities were considered for finding a good match. In the current step, the entire case, including relationships, attributes, and other properties of the base case, is considered for adaptation. Figure 6 is a flowchart of this step.

Entity Adaptation: Entities are adapted in a sequence of three steps. First, all entity types that have an index value of "y" for the base case but an "n" for the new case, along with their attributes and relationships, are dropped from further consideration.
Figure 6. Base Case Adaptation and Evaluation in CABSYDD
Second, all root-level entity types with an index value of "y" for both the base and the new cases will be considered one by one. Third, the richest existing root-level entity type is considered next. "Richest" is defined as the one with the most subclasses. The rationale is based on the heuristic that the richer a chunk of entity types is, the more important it is in a case.

Example: Consider the adaptation of case 3 for case 4. No entities are dropped from further consideration, because none has an index value of "y" for the base case and an "n" for the new case. Two root-level entity types have an index value of "y" for both the base and the new cases. These are STUDY ROOM and LOCKER. In case 3, the root-level entity type HOLDING, with four subclass entity types, has the most subclasses. Therefore, it will be considered before USER, which has three subclasses.

As suggested by Storey and Goldstein [44], the system checks for hidden entities and relationships. A hidden relationship is discovered when the name of a nonkey attribute is identical to either an entity name or the key attribute of an entity. In that case, such nonkey attributes are removed and the relationship is noted for later naming and definition in the new case. Similarly, a potential indication of the existence of a hidden entity is that two nonkey attributes of two different entities have the same name. In such cases, CABSYYY asks the designer for confirmation of the existence of this entity.

Transformation of a chunk of entity types from the base case follows a top-down procedure. If the new case does not have a similar entity type in the base case, then the whole chunk is dropped from further consideration. The rationale is that a root-level entity type is more general than its subtypes. It is more efficient to compare the general properties than the more specific ones. If the root-level entity type is deemed to be useful, then transformation begins. The root-level entity type’s name, role, and domain can be modified to suit the new case. Next, the components of the key attribute of the base case are considered one by one. For each acceptable one, the attribute name is adapted. If a key attribute is not transferred, it will be reconsidered for adaptation as a nonkey attribute later. Any new key attribute needed but not available from the base-case entity type is defined by the user. Next, all nonkey attributes are considered one by one. For each, the attribute can be accepted or rejected. New nonkey attributes that are not available from the base entity type are defined by the user.

Next, all applicable clusters under the root entity are considered. Since each cluster must be based on a defining attribute of the superclass entity type, any base-case cluster that is based on a dropped defining attribute is omitted from consideration. New subclasses, not available from the base-case cluster, can be defined by the user. Similarly, after all the base-case clusters have been considered, new clusters can be defined.

Following the definition of the first-level subclass clusters of a root-level entity type, the transformation of the next lower level begins. After all the base-case entity type chunks have been considered, new ones can be defined.
Example: In adapting case 3 for case 4, the defining attribute H-TYPE has been transferred. This cluster has four subclass entity types: BOOK-CIR, REFERENCE, PERIODICAL, and VIDEO. The new case, case 4, only needs BOOK-CIR and REFERENCE. Therefore, VIDEO and PERIODICAL will be dropped. In addition, a new root-level entity type, SPECIAL-PROGRAM, is defined for case 4.

Relationship Adaptation: Relationships are adapted in two stages. First, the existing transferable relationship types are adapted. Second, new relationship types are discovered through foreign keys of the defined entity types. Base-case relationship types whose related entity types have been transferred are considered next. A base-case relationship type can be redefined to be a relationship type between any two entities within a chunk. Existing relationship type attributes can be transferred if applicable.

Example: In adapting case 3 for case 4, since entity types PERIODICAL and VIDEO are not transferred, the relationship types CHECKS-OUT-P and CHECKS-OUT-V are dropped. CHECKS-OUT-B and CHECKS-OUT-R are examples of the relationships that will be considered for adaptation because their related entities were adapted earlier. The relationship type STU-ASSIGNED-L in the base case is a relationship between STUDENT and LOCKER with two attributes. For case 4, it is transformed to a relationship between USER and LOCKER called ASSIGNED-L with one attribute. The relationship type STA-ASSIGNED-L between LOCKER and STAFF in the base case is dropped for the new case. The relationship JOINS between USER and SPECIAL-PROGRAM in the new case is defined by using first principles of knowledge.

Evaluation: After all the entity and relationship types are defined for the new case, overall validity checks are performed. These include checking for the existence of dangling entity types and unnecessary subclasses. An entity type chunk is dangling if none of its entity types participates in a relationship type. This is a violation of a well-formed conceptual schema. The designer must define the necessary relationship types to link the chunk to at least one other entity type in the schema. An entity type is unnecessary if it does not have any attribute, does not participate in a relationship, or does not have clusters of subclasses of its own.

Step 4—Learning: A high-level flowchart of CABSYYDD’s learning algorithm is shown in Figure 7. The theory of learning from observation was used in CABSYYDD. This theory is a general form of inductive reasoning that includes the creation of classification criteria to form taxonomic hierarchies without the benefit of an external teacher [8, 17]. In passive observation, a learner classifies observations of multiple aspects of the environment. Learning from observing significant events leads to the generalization of the new events into a new concept or the discovery of new domain concepts and relationships [17]. In CABSYYDD, the decision to add a new case and the justification for doing so depend on the observed structural differences between the new and the existing cases. (See the example presented using Figure 5.) The dif-
ferences between a base case and the new case will be generalized to become new indices in the case base. No user intervention is needed. In order to prevent uncontrolled growth of new indices and cases, only selected features are chosen for creation of new learned indices by the system.
The very first case for a department, which is either defined from the first principles or derived from a neighboring industry group, does not have any learned indices. This is because there is no other case for the department to which it could be compared. A second case is added only if it is the source for one or more newly created learned indices. The creation of a different root-level entity type results in the creation of a new learned index. The entity type name of a base-case root-level entity type that has not been transferred becomes the name of a new index. For this index, all existing cases will have an index value of “y” and the new case will have an index value of “n.” Similarly, the entity type name for a new root-level entity type that was defined from first principles will become the name of a new index. For this index, all existing cases have a value of “n” and the new case has a value of “y.” It is important to note that design and learning are distinct from each other. The two are not iterative. A new case must be completely designed before learning from it occurs.

Implementation

As part of this research, we constructed two systems. First, SYDD contains first principles knowledge of database design. It does not learn from experience. Second, CABSYDD contains case-based reasoning logic as well as the case base itself. It learns from experience. Both prototypes were implemented in CLIPS Version 6.0. (The latest version of CLIPS can be found in [15].) SYDD assists in designing from first principles when no similar cases for the new problem exist in the case base. CABSYDD takes over when a similar case exists. Furthermore, SYDD is invoked by CABSYDD to assist in designing those parts of the problem for which there are no corresponding constructs in the base case.

The architecture of CABSYDD is shown in Figure 8. There are six major components: Knowledge Base, Case Base, Inference Engine, Working Memory, User Interface, and Help Module. The first two will be described in detail below. The Inference Engine is data driven. For design problems, data-driven methodologies are superior to goal-driven ones (see [6]). The rationale is that there is no specific goal to be reached. The design, which is the goal, is unknown ahead of time. Hence, a goal-driven approach is not suitable. The Working Memory is used to retain dynamic information that ties a base case to the new case. The User Interface is in the form of system-initiated dialogues. The Help Module provides context-sensitive help.

The case base has two major components. These are case-base indices and a set of cases. Indices are implemented in the ordered facts of CLIPS. For example, information on a “sector” of the NAICS [33], as shown in Figure 4, is modeled as: (sector 61 educational services 1). This reads as follows: “Sector number 61 is called ‘educational services’ and it has one subsector.” Another example is the following: (subsector 61 611 educational services 7). It represents that under sector number 61 there is a subsector number 611 called “educational services” and it has 7 industry groups.

An entity type is coded in a template of CLIPS. As an example, consider the following, which represents the entity USER from Appendix A:
Figure 8. The Architecture of CABSYDD

```
(old_et (et_name USER)
         (et_role "MEMBERS OF THE LIBRARY")
         (et_domain "ALL MEMBERS")
         (no_key_attr 1)
         (no_nonk_attr 3)
         (at_level 1)
         (chunk_rt YES)
         (totdescs 3)
         (totclusters 1)
         (immed_sup USER)
         (allsup USER))
```
USER’s role and its domain descriptions are in double quotes. The slot “no_key_attr” specifies the number of key attributes (which is one here). The slot “no_nonk_attr” specifies the total number of nonkey attributes (which is three here). The slot “at_level” describes the position of the entity type in the chunk. USER happens to be a root-level entity type, hence a value of one. Slot “chunk_rt” serves as a flag to indicate whether any relationship type is involved in any of the entity types in the chunk. This slot is only used by root-level entity types for detecting dangling entity type chunks. Every root-level entity type has a default value of “NO” for this slot. When an entity type of this chunk becomes a participating entity type in a relationship type, the value is switched to “YES.” During validity checking, any root-level entity type that still has a value of “NO” is flagged as dangling. Slot “totdescs” specifies the total number of subclasses in the chunk. It represents the richness of the chunk. Slot “totclusters” records the total number of immediate subclass clusters under an entity type. Slot “immed_sup” specifies the immediate superclass entity type. Slot “allsups” is a list of all the superclasses, bottom up, to the root-level entity type.

The knowledge base contains about 1,000 rules. As shown in Figure 8, the rules are divided according to the stages of performing different tasks at different times.

**Stage 0:** The rules of this stage are activated when a design session ends. The case-base directory is updated. All housekeeping chores are done during this stage.

**Stage 1:** The rules are used to search for a base case. The search is hierarchical from Sector to a specific Industry Group.

**Stage 2:** The rules are used to search for the best base case. The one that is the most similar to the new case is chosen from among candidates.

Altogether, there are 73 rules in stages 0, 1, and 2.

**Stage 3:** This stage is activated when it is decided that the design of an entity chunk should proceed from the first principles. A total of 126 rules are used in this stage.

**Stage 4:** This stage is invoked when a relationship type is to be defined from the first principles; 149 rules exist for this stage. This stage starts by presenting to the designer hidden relationship types, if any, which were discovered during stage 3. Following that, new relationship types are defined. During this stage, overall validity of the schema is checked and corrections are made if needed.

**Stage 5:** During this stage, with the help of 358 rules, entity type chunks are defined either through adaptation or from first principles.

**Stage 6:** Relationship types are defined during this stage. There are 139 rules. Relationship types of the base case are considered first. Then any hidden relationship types that were discovered in stage 5 are defined. The definition of new relationship types follows. Finally, similar to stage 4, overall validity checking is performed.
Stage 7: With the help of 136 rules, new learned indices are created during this stage.

From data modeling and case-based reasoning perspectives, the rules can be classified into two major classes. First, data modeling rules are further subclassified into three groups. "Model Construct" rules contain the EER model rules. These are used by SYDD for design from first principles. Most were derived from Elmasri and Navathe [20]. “Procedural” rules embody the step-by-step methodology for conceptual design from first principles. These were derived from Chen and Associates [10], Storey et al. [45], and Teorey [48]. “Validation” rules are activated in stages 3, 4, and 6 to check for consistency and completeness. Second are the rules on case-based searching, retrieval, adaptation, and learning. These are original to this work. They are activated in stages 1, 2, 5, 6, and 7.

A pseudocode of one of the rules of stage 5 is provided in Figure 9. Through this rule, CABSYYD chooses an existing entity type chunk (from a selected base case) to be adapted for a new case. The root-level entity type of this chunk is a learned index in the case base. Both the base and the new case have the same index value of “YES” for the root-level entity type. CABSYYD displays the entity type name, role, and domain to the user, and proceeds with adaptation using other rules (which are not shown here).

Evaluation of the Systems

As is pointed out by Tichy [49], the development of a novel computer system is not per se sufficient proof of its scientific merit(s). The system must be subjected to empirical evaluation. For knowledge-based systems such as CABSYYD, the evaluation includes the validation of system outcome and verification of system utility or effectiveness [55]. The evaluation approach used here was similar to the one used by Vinze [53]. We present details of the validation and verification of the systems in [11]. A summary is presented here.

Two expert human designers were employed to validate both systems. Three database course project problems were used for CABSYYD and two for SYDD. For each problem, the experts were given complete user requirements in English and the output schemata from each system. The experts were then asked to comment on each schema. After some clarification and discussion, both agreed that the schemata are technically correct and do not violate any conceptual design principles.

Verification was carried out with an experiment using 31 undergraduate students as subjects. The objectives were to compare the effectiveness of case-based design by CABSYYD to that of first principles by SYDD. A major finding was that the two systems did not differ in their level of perceived effectiveness. However, when it came to the bottom line of choosing one of the two, there was a statistically significant preference among the subjects for using CABSYYD.

From the productivity perspective, the average number of design errors per subject per output schema was less for CABSYYD (1.33) than for SYDD (4.33). However,
If
1. the system is in stage 5, and
2. the system should be adapting entity types, and
3. the system should be checking learned indices for which both the base case and the new case have the same index value, and
4. the system should be choosing the next entity type chunk to adapt, and
5. there is such a learned index for which both the base case and the new case have a value of “yes,” and this index has not yet been considered for entity type adaptation, and
6. there is such an entity type in the base case that both (a) serves as that index and (b) has not been considered for adaptation yet,
then
7. display the entity type name for the user, and
8. display its role in the base case, and
9. display its domain in the base case, and
10. display two messages (a and b) that this entity type and the whole chunk will be chosen for adaptation next, and
11. remove the fact that this index has not been considered and also remove the fact that the system should be choosing the next entity type chunk to adapt, and
12. set up this entity type and the whole chunk as the next construct to be adapted.

Figure 9. An Example of a Rule

an average design session with CABSYYD took 17.8 minutes, whereas a session with SYDD took 10.8 minutes. The former took longer because the subjects had to read information on every base case presented to them by CABSYYD and then make a decision to accept or reject it for adaptation. We believe that even if more time had been spent on the design from the first principles, case-based reasoning would have resulted in a better design, as reflected in one subject’s comment: “The case-based approach requires one to think more in depth than the other approach [design from the first principles].” The lower average number of errors for CABSYYD strongly supports this opinion.

We firmly believe that tools such as CABSYYD should be empirically evaluated. Eventually, strengths of several empirically proven prototypes (such as CABSYYD) will be merged together by vendors to create commercial tools.

Comparison to Related Works

WE HAVE FOUND TWO OTHER SYSTEMS that use variations of case-based reasoning for database design. These are CSBR and DES-DS, which are reported in Storey et al. [46] and Paek et al. [36], respectively. The comparison below is based on what is
reported in these two citations. Table 2 is a summary of our comparison between these and CABSYDD.

CABSYDD gathers input by interacting with a designer. Input to CSBR is an Entity Relationship Model (ERM) that is generated by an expert database design system called View Creation System (VCS) [44]. VCS performs a similar function as that of SYDD. However, it is not tightly integrated with CSBR. SYDD and CABSYDD are tightly integrated. More importantly, in CABSYDD, the use of case-based reasoning or reasoning from first principles is transparent to the user. In CSBR, a user must go through VCS’s reasoning from first principles to generate an ERM. The output is then fed into CSBR for analysis and modification. A user would potentially go through several iterations between the two systems to arrive at a final ERM. CABSYDD’s case-based reasoning rules (to generate an ERM) are complete and self-contained.

DES-DS borrows from Choobineh et al. [13, 14] in restricting its input to forms and reports. Its output is a set of third normal form relations. By comparison, similar to CSBR, CABSYDD’s input is not restricted to forms and reports, and its output, like the majority of the modern design tools, is an ERM. An ERM is closer to the rich semantics that are encountered in requirements identification and elicitation than the relational model. Furthermore, translation from an ERM to the relational model is algorithmic.

CABSYDD is based on a complete and unmodified adaptation of NAICS [33]. This has the advantage of being based on a rigorous, standardized, complete, and widely recognized classification of businesses in North America. CSBR uses an adaptation of part of NAICS. DES-ES is not based on such a classification system.

Although a prototype, CABSYDD’s implementation is completed. At the time of their published reports, both Paek et al. [36] and Storey et al. [46] explained that their implementations were not complete. In addition, Paek et al. [36] reported that EDS-DS was based on a single domain. CABSYDD and CSBR are domain independent. Paek et al. [36] do not report on tests of validity, effectiveness, or usage productivity. Storey et al. [46] report on validation tests of CSBR, but not on its effectiveness or effect on productivity. As presented earlier, we conducted validation tests on CABSYDD, as well as laboratory tests on human subjects to assess CABSYDD’s effectiveness and measure its productivity relative to design from first principles of SYDD.

One part of the results from our empirical tests—that is, correctness of design using a knowledge-based system—is supported by a similar study. Batra and Antony [2] compared two groups of subjects. One group used an intelligent tool called CODA that is endowed with first principles knowledge, similar to SYDD. The other group used a system with a similar “look and feel,” but without the expertise contained in CODA. Similar to our experiment, the independent variables were the two types of systems and two types of tasks. The control variable was the same “look and feel” of both systems. The dependent variable was the performance of the designers measured by correctness of design. Covariate was subjects’ prior knowledge. The result of the study was that CODA benefited subjects with lower prior knowledge more than those with higher prior knowledge. That is, it was more useful to those who needed assistance.
Table 2. Case-Based Database Design Systems—A Comparison

<table>
<thead>
<tr>
<th></th>
<th>CABSYDD</th>
<th>CSBR [46]</th>
<th>DES-DS [36]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source of input</td>
<td>System interacts with a designer.</td>
<td>ERM output from another system.</td>
<td>Designer provides description of a form or a report.</td>
</tr>
<tr>
<td>Process and output</td>
<td>Creation of an ERM including consistency and completeness checks.</td>
<td>Analysis and modification of an ERM including consistency and completeness checks.</td>
<td>Creation of third normal form relations from forms and reports.</td>
</tr>
<tr>
<td>Required number of steps in usage</td>
<td>One.</td>
<td>At least two.</td>
<td>One.</td>
</tr>
<tr>
<td>Basis for the case base</td>
<td>Based on complete and unmodified adaptation of NAICS.</td>
<td>Based on a partial and modified adaptation of an earlier version of NAICS.</td>
<td>No basis exists. Cases are created, organized, and used later.</td>
</tr>
<tr>
<td>Implementation completeness</td>
<td>Yes.</td>
<td>No [46, p. 474].</td>
<td>No [36, p. 94].</td>
</tr>
<tr>
<td>Domain independence</td>
<td>Domain independent.</td>
<td>Domain independent.</td>
<td>Single domain.</td>
</tr>
<tr>
<td>Testing</td>
<td>Validation of approach, effectiveness of usage, and measures of productivity.</td>
<td>Validation of approach.</td>
<td>None.</td>
</tr>
</tbody>
</table>
Summary, Conclusions, and Future Directions

WE DESCRIBED THE THEORY, DESIGN, AND CONSTRUCTION of a case-based reasoning system for conceptual database design. CABSYDD’s customized indexing method for conceptual design—its learning algorithm, domain independence, as well as its complete and self-contained knowledge base—represents a step forward in machine learning for database design. The empirical experiment indicated that the approach in CABSYDD not only is preferred over the design from first principles but also results in fewer errors by the subjects. In short, this research, development, and experimentation effort provides additional support for the utility of case-based reasoning in database design.

The unique contributions of the research reported in this paper include: (1) implementation of the extended ERM as well as four other constructs to facilitate case-based reasoning for conceptual database design, (2) unique indexing and learning method, and (3) discovery and construction of rules for case-based searching, retrieval, adaptation, and learning for conceptual database design. The immediate beneficiaries of this research will be firms that are engaged in the development of tools for information systems design and construction. The eventual beneficiaries will be database designers. With a commercial version of such a database design tool, designers will be more productive. Such a system will especially be useful to the database designers of software consulting firms who are engaged by various clients. Past design experiences will be reused for future clients in the same or similar industries. As a result, these kinds of firms will have the potential to realize gains in productivity and therefore competitiveness.

Future research direction includes enhancement of CABSYDD with an ontology for the NAICS [33]. An ontology is a formal and explicit definition of the concepts and the relationships among these concepts [42]. Because they are formal definitions, the content of an ontology can be directly used for inference by a computer program to generate related new knowledge or to make a judgment about a certain statement from a human user. Recall that, in developing CABSYDD, NAICS was used as the high-level classification of industries. Although there have been several efforts to develop ontologies for database design tools, including those reported in Storey [43], Storey et al. [45, 46], Ulrich et al. [52], and Wand et al. [54], to the best of our knowledge, currently there is no ontology for the whole of NAICS. In CABSYDD, the domain knowledge of the departments within industries is mostly dispersed or partially repeated in various individual cases. The cases are classified and stored in the adaptive self-learning case base. In an ontology of NAICS, the prototypical knowledge of industries and departments within each is captured. Using this knowledge, the design tool can make better and more intelligent comparisons of existing cases with a new case. To enhance interoperability and enable import or export of another ontology, such that the knowledge and inference is tremendously magnified, this ontology must be coded in a mainstream ontology language, such as knowledge interchange format (KIF) [27] or ontology inference layer (OIL) [35].
NOTES

1. This notion of domain should not be confused with at least two others. One is the traditional definition of domain as a named collection of values of an attribute. Column heading “Attribute Domain” in Figure 2b is an example of the use of this definition. The other definition is the one given in Navathe et al., where domain is defined as “all instances of the class at some given time” [32, p. 52]. In our context, “domain” is used to assist a human designer in deciding to choose entities for adaptation.

2. See Glass and Vessey [23] for a discussion of taxonomies for application development.

REFERENCES


17. Dejong, G. An approach to learning from observation. In R.S. Michalski, J.G. Carbonell,


42. Smith, B. Ontology and information systems. Department of Philosophy, State University of New York at Buffalo (available at ontology.buffalo.edu/smith/articles/ontologies.htm, last visited November 14, 2004).


Appendix A: Details of Case 3

THE FOLLOWING IS THE DESCRIPTION of the requirements of a college library.

Values of Predefined Indices

Sector: educational services
Subsector: educational services
Industry group: colleges, universities, and professional schools
Department: library

Case Description

The library wants to keep track of its library holdings and users. Holdings include (1) books for circulation, (2) books in the reference section, (3) periodicals in the periodical room, and (4) videos.

Holdings are identified by call numbers. Call numbers are strings. There are four types of holdings: books, reference items, periodicals, and videos. A book for circulation has a call number, a title, and 1 to 3 author names. An item in the reference section has a call number, a title, and a publisher name. A periodical has a call number, a title, a current volume, and a current number. The library does not have a separate call number for each issue of a periodical. A video has a call number, a title, and a format. Possible values for the format are "VHS" and "DVD."

A library user has exactly one social security number, a name, an address, and a user type. Possible user type values are "STUDENT," "STAFF," or "COMMUNITY-USER." A community user has a membership expiration date. Each user can only assume one type value at any one point in time.

A user can check out any number of books for circulation at any one point in time, and a due date has to be recorded for each loan. A user can check out at most three books in Reference. The due date and due time are recorded. A user can check out at most five different periodical issues at a time. Due date, due time, volume, and number for each issue are recorded. A user can check out at most two videos at the same time with a due date.

The library provides study rooms and lockers to its users. A study room has a study room number and location (all strings). A STUDENT user can be assigned to a study room. Each assignment has a start date and expiration date. A student user may be assigned to one and only one study room. Each study room, if assigned, is assigned to at most one student at the same time. In other words, no two students can share a study room at any point in time. A locker has a locker number (string), location (string), and size code (string). A STUDENT or STAFF user can be assigned to at most one locker. Each locker, if assigned, is exclusively assigned to one user. Each assignment includes a start date and an expiration date.
Appendix B: Details of Case 4

THE FOLLOWING IS THE DESCRIPTION of the requirements of a private, small, liberal arts college library. Its entity-relationship diagram (ERD) is shown in Figure B1.

Values of Predefined Indices

Sector: educational services
Subsector: educational services
Industry group: colleges, universities, and professional schools
Department: library

Case Description

The library wants to keep track of its library holdings, users, special programs, lockers, and study rooms for its library users.

Holdings of the library are of two types: books and reference items. A book for circulation has a numeric (integers only) call number, a book title, and 0 to 3 author names. An item in the reference section has a numeric (integers only) call number and a title.

A library user has exactly one social security number, one name, one address, and a user type. There are two kinds of users: STUDENT and STAFF.

A user may check out any number of books for circulation. The due date has to be recorded. A user can check out at most two items from the reference section. Due date and due time are recorded for reference items.

The library provides special programs to its users. Each special program has a program code, a program name, a program start date, and a program end date. Any user can join any number of special programs.

The library offers lockers to its users. A locker has a locker number (numeric) and location (string). Any user can be assigned to a locker. Each assignment includes an expiration date. The university does not allow a user to have more than one locker.

Another offering of the library is the study rooms. A study room has a study room number (numeric) and location (string). Only a STUDENT user can be assigned to a study room. Each assignment has a start date and expiration date. A student, if assigned to a study room, can be assigned only throughout the assignment. Each study room, if assigned to students, is assigned to at most two students at the same time. In other words, two students can share a study room at any point in time.
Figure B1. The EER Diagram for Case 4

The following Statement of Ownership, Management, and Circulation is provided in accordance with the requirements contained in 39 USC 3685. Journal of Management Information Systems (Publication no.0742-1222) appears quarterly and the annual subscription price is $569.00. It is owned, managed, and published by M.E. Sharpe, Inc., which is located at 80 Business Park Drive, Armonk, Westchester County, NY 10504-1715. The Publisher is M.E. Sharpe, at the same address. The Editor is Vladimir Zwass, c/o M.E. Sharpe, Inc. 80 Business Park Drive, Armonk, NY, 10504. During the preceding twelve months the average number of copies printed for each issue was 1,250; the average paid circulation (by mail subscription) was 864; the average free distribution, 48; the average number of copies distributed, 912. Corresponding figures for the issue published nearest to the filing date: total number of copies printed, 1,250; total paid circulation (by mail subscription), 966; total free distribution, 75; total distribution, 1,041. Filed by Vincent Fuentes, Senior Vice President.