DBE: AN EXPERT TOOL FOR DATABASE DESIGN

Dina Bitton *
Jeffrey C. Millman *
Solveig Torgersen +

Department of Electrical Engineering and Computer Science
University of Illinois at Chicago

1. Introduction

DBE (Design By Example) is an expert tool for specifying and designing a relational database and its associated queries. The primary goals in the development of *DBE* were:

(a) To provide a system that can tolerate, elucidate, and correct poorly formed relational database specifications by using example tables and example queries.
(b) To integrate the design of a database with the design of its queries.
(c) To construct well-formed database schemes by automating important knowledge in relational design theory.
(d) To study issues of feasibility, ease of use, and performance in automatic database design.

Currently, a prototype of *DBE* is operational on Sun 3 workstations and interfaces with SunIngres. The system supports an interactive design process, which takes the database designer through a structured sequence of *Design Screens.* The screens visualize

* Author's present address: DB Software Corporation, 385 Sherman Avenue Suite 2, Palo Alto CA 94306, USA.
+ Author's present address: Norsk Hydro a.s., Postboks 200, N-1321 Stabekk, Norway.
Partial Support for this research was provided by a fellowship from the Norwegian Council for Sciences and Humanities.

design information and provide menu selections to invoke specialized editors and design algorithms. An underlying storage subsystem provides access to a design catalog containing the attributes, dependencies, relations, tables, and queries participating in a design session. Throughout the design process, a dual representation of relations and queries is available to the designer. One is the graphical *DBE* representation, the other is text in a conventional query language (a Quel query in the present implementation).

A design session with *DBE* establishes a dialogue between the database designer and the system, during which the designer can convey his knowledge about the semantics of the database through the use of *example tables* and *example queries*. Visual tools, such as specialized editors, menus, and graphical displays of the examples, provide an interface that hides the complexity of the design algorithms used by the system. This interface allows the designer to focus on specifying the application, instead of trying to master concepts in relational design theory.

The design expertise embedded in *DBE* addresses two important problems that current database design tools often overlook. The first problem is that the database designer is usually not able to specify the semantics of the database or the queries correctly and completely in one step. Thus a good design tool should support an iterative design process and provide a way to verify and correct an initial specification. For that purpose, *DBE* represents design information with examples that the designer can examine and modify. The second problem is that alternative database schemes often exist of which the designer is not aware. During a conventional design process, one out of possibly many candidate schemes is often selected, without weighting whether it is natural for representing data or for formulating queries. In contrast, *DBE* generates multiple designs for the same database and lets the designer examine and evaluate them in conjunction with queries.

Anomalies resulting from an incorrect specification of the logical dependencies often appear in a relational scheme that is produced by a conventional design tool or chosen by the designer in an ad-hoc manner. Even for the best understood dependencies, the functional dependencies, omissions or errors will often occur when the database designer is asked to specify them all at once. To cope with this problem, *DBE* can automatically infer functional dependencies from an example table containing sample data. Alternatively, if the designer prefers to specify the dependencies directly, but needs to verify this specification, *DBE* will generate an example table which satisfies exactly these dependencies. The data values used in generating the example table are selected from domains defined by the database designer. Thus by examining the familiar data in the table, the designer may uncover inconsistencies. In particular, for any dependency that was omitted in the initial specification, the example table will contain a pair of "real-like" tuples violating this dependency.

Other database anomalies may result from the designer choosing a scheme that is not sufficiently normalized or does not have other desirable properties such as lossless joins or preservation of dependencies [Ul82]. To avoid these anomalies, *DBE* generates database schemes satisfying these properties, or checks whether these properties hold for a scheme which was manually constructed by the designer.

Ambiguities in a relational query may result from an unnatural database scheme, from an unfriendly query language, or simply from a user error. Often, these ambiguities could have been avoided by making the database designer aware of an alternative scheme for the database, and by providing a tool for validating the formulation of queries.

DBE provides the database designer with the option to reject or modify a candidate database scheme, if it is difficult to formulate a desirable set of queries against that scheme. Thus the design and acceptance of a database scheme is fully integrated with the design of the queries. Furthermore, the design tool can be used on an operational database to update and modify the database scheme if new data and new applications make the existing scheme incomplete or inefficient.

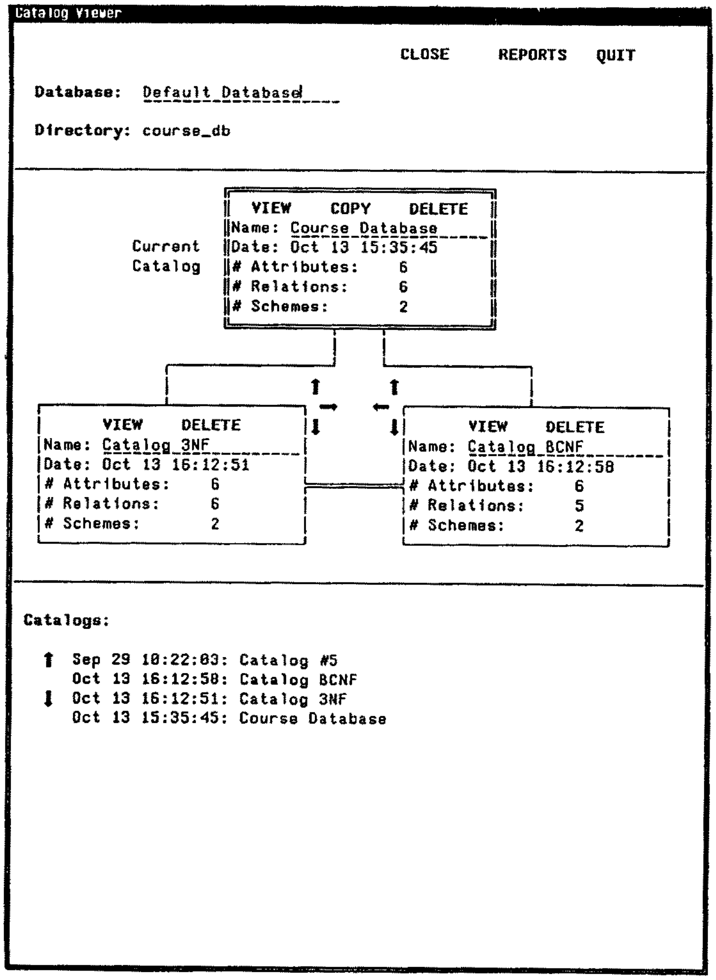
The remainder of this paper is organized as follows. In Section 2, we present the *Schema Design Screens* and the design algorithms utilized in the database design process. In Section 3, we describe the *Query Design Screen.* In Section 4 we provide an overview of the data structures that constitute the *DBE* design catalog, and describe the screen editors associated with these structures. In Section 5, we present the storage subsystem and discuss performance issues. Section 6 contains a summary and outlines directions of future work.

2. The Schema Design Screens

A design session with *DBE* takes the database designer through a sequence of design screens, where current design information is displayed and menu selections are provided. Help is available in pop-up windows, and a transparent history mechanism keeps track of the work accomplished during the session. Certain screens are *Viewers,* from which specialized editors can be invoked. For instance, there is a *Catalog Viewer* (Figure 1) and a *Scheme Viewer.* Two other screens, the *Relation Viewer* (Figure 3) and the *Decomposition Viewer* (Figure 4), which encompass the most important functions and algorithms utilized during the design process, are described below.

2.1. The Relation Viewer

FIGURE 1 - CATALOG VIEWER



Database: Default Database

Directory: course_db

Current Catalog

Name: Course Database
Date: Oct 13 15:35:45
Attributes: 6
Relations: 6
Schemes: 2

Name: Catalog 3NF
Date: Oct 13 16:12:51
Attributes: 6
Relations: 6
Schemes: 2

Name: Catalog BCNF
Date: Oct 13 16:12:58
Attributes: 6
Relations: 5
Schemes: 2

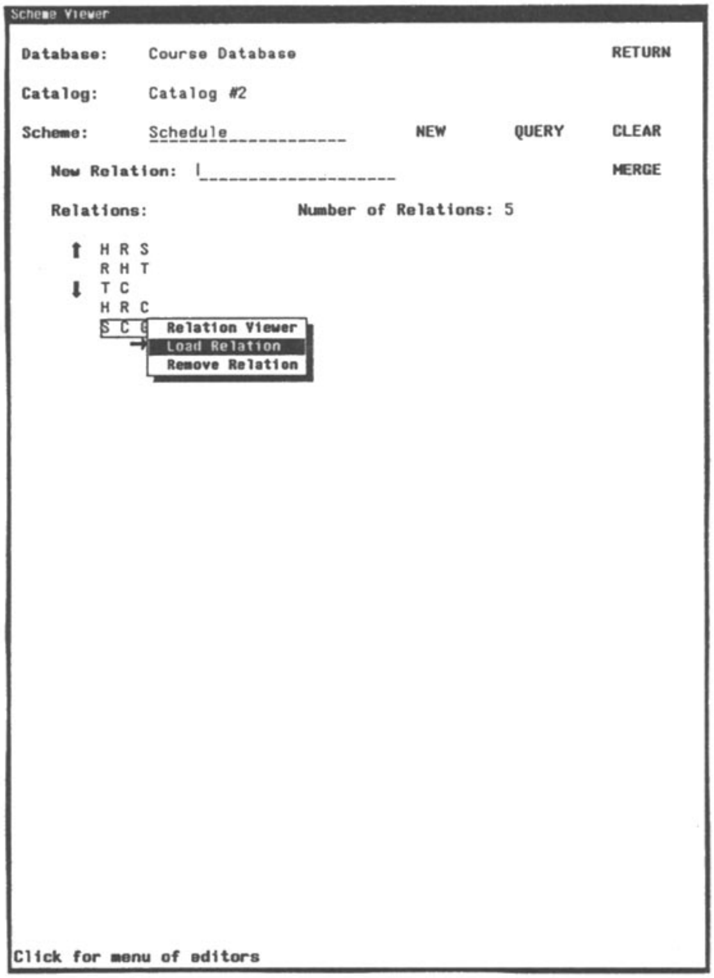
Catalogs:

```
  Sep 29 10:22:03: Catalog #5
  Oct 13 16:12:58: Catalog BCNF
  Oct 13 16:12:51: Catalog 3NF
  Oct 13 15:35:45: Course Database
```

This screen displays a complete description of a relation (Figure 3). It is divided into four areas, each displaying one type of information on the relation: (1) keys, (2) attributes, (3) dependencies, and (4) example table. In each area, the current information stored in the catalog is displayed, and can be edited by selecting the corresponding editor.

Two other menu selections, *SHOW EXAMPLE* and *INFER DEPENDENCIES*,

FIGURE 2 - SCHEME VIEWER

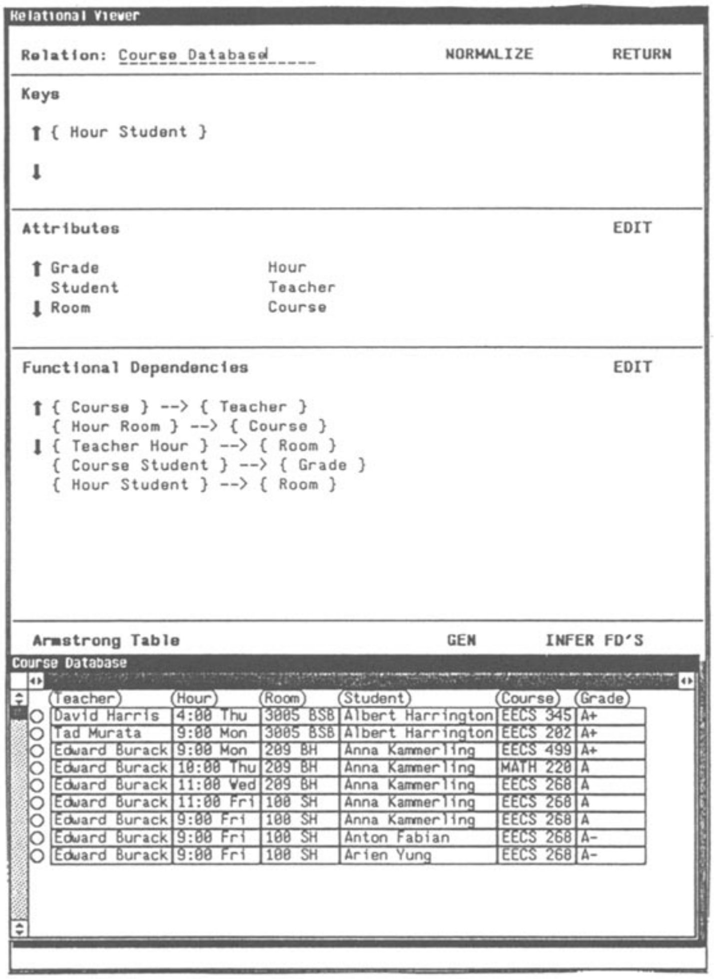


```
Scheme Viewer

Database:     Course Database                                RETURN

Catalog:      Catalog #2

Scheme:       Schedule____________      NEW        QUERY     CLEAR

   New Relation:  |___________________                       MERGE

   Relations:               Number of Relations: 5

     ↑  H R S
        R H T
     ↓  T C
        H R C
        S C G  Relation Viewer
               Load Relation
               Remove Relation

Click for menu of editors
```

should be explained in more detail, since they control the use of example tables in the design process. The designer may provide an example table, or request that one be generated by the system by selecting the *SHOW EXAMPLE* menu selection. The example table generated by *DBE* is always an Armstrong table [Ar74, BDFS84]. That is an instance of the relation which *exactly* satisfies the given set of dependencies.[1] In

[1] The idea of using Armstrong tables in the design process was originally proposed in [SM81]. An algorithm for constructing an Armstrong table was given in [BDFS84]. Bounds for the size of an Armstrong table and an improved algorithm, which we have implemented in

FIGURE 3 - RELATION VIEWER



Relational Viewer

Relation: Course Database

NORMALIZE RETURN

Keys

{ Hour Student }

Attributes EDIT

Grade	Hour
Student	Teacher
Room	Course

Functional Dependencies EDIT

```
{ Course } --> { Teacher }
{ Hour Room } --> { Course }
{ Teacher Hour } --> { Room }
{ Course Student } --> { Grade }
{ Hour Student } --> { Room }
```

Armstrong Table GEN INFER FD'S

Course Database

Teacher	Hour	Room	Student	Course	Grade
David Harris	4:00 Thu	3005 BSB	Albert Harrington	EECS 345	A+
Tad Murata	9:00 Mon	3005 BSB	Albert Harrington	EECS 202	A+
Edward Burack	9:00 Mon	209 BH	Anna Kammerling	EECS 499	A+
Edward Burack	10:00 Thu	209 BH	Anna Kammerling	MATH 220	A
Edward Burack	11:00 Wed	209 BH	Anna Kammerling	EECS 268	A
Edward Burack	11:00 Fri	100 SH	Anna Kammerling	EECS 268	A
Edward Burack	9:00 Fri	100 SH	Anna Kammerling	EECS 268	A
Edward Burack	9:00 Fri	100 SH	Anton Fabian	EECS 268	A-
Edward Burack	9:00 Fri	100 SH	Arien Yung	EECS 268	A-

both cases, the example table can be edited by invoking the table editor, and dependencies will be inferred which reflect any update to the values and tuples stored in the example. For instance, changing the value of an attribute involved in a previously

DBE, are given in [MR86].

defined dependency may make the table violate this dependency. Thus the dependency should be deleted from the set associated with the relation. After the example table is updated, the *INFER DEPENDENCIES* menu can be selected. This selection will have the effect of updating the set of dependencies so that it reflects the changes made by the designer in the example table. After any update of the attributes or dependencies, the keys and the example table are automatically updated.

Four algorithms provide the functionality associated with the Relation Screen:

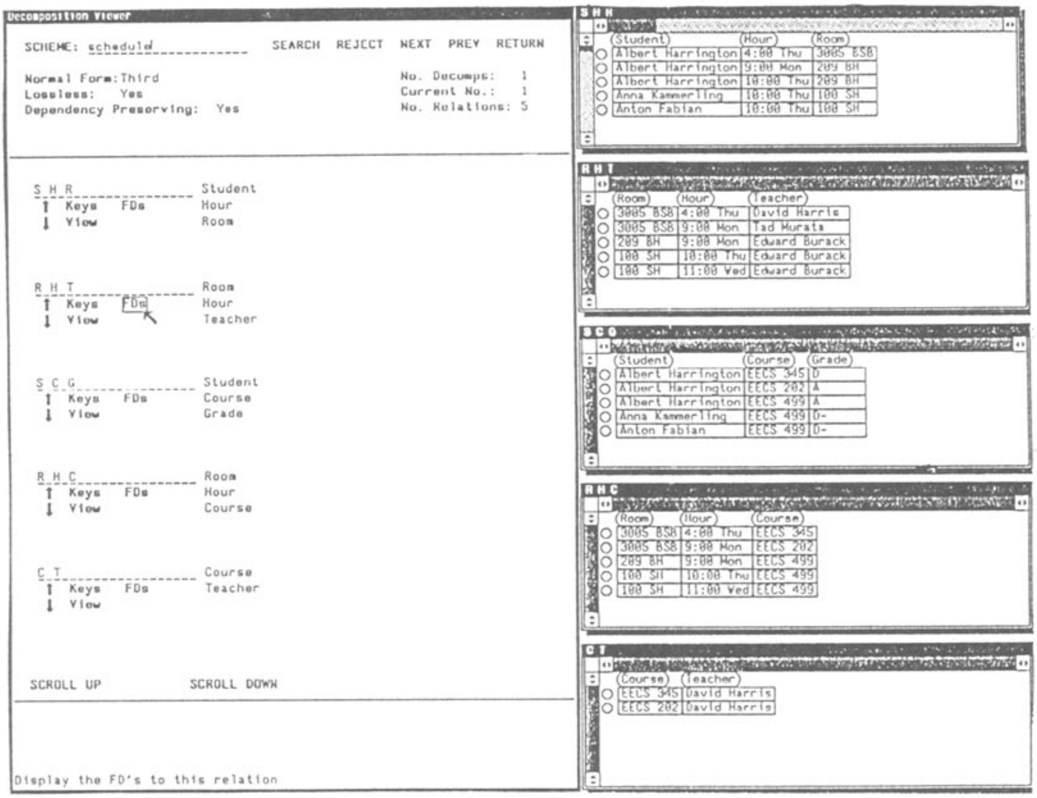
- reduction of a set of functional dependencies [Ul82]
- finding keys, given a set of functional dependencies [LO78]
- generating an Armstrong table for a set of functional dependencies
- inferring the functional dependencies satisfied by a table [MR87, BMOT87].

2.2. The Decomposition Viewer

This screen (Figure 4) will appear after the designer selects the *NORMALIZE* menu item in the Relation Screen. It contains a scheme (set of subrelations) obtained by decomposing a relation into third, Boyce-Codd, or fourth normal form, and a label describing properties of the decomposition: normal form, preservation of dependencies, lossless join. Many alternative decompositions may exist for a given relation. When the normalization function is invoked, the first scheme found by the decomposition algorithm is generated and displayed. If the designer wishes to examine alternative schemes, these can be displayed by selecting the *NEXT SCHEME* option. We use an algorithm proposed in [To87] to efficiently generate all the distinct BCNF decompositions.

The relations produced by decomposition inherit functional dependencies obtained by projecting the set associated with the original relation. The decomposed scheme, with the relations and the projected dependencies, are automatically stored in the

FIGURE 4 - DECOMPOSITION VIEWER



```
SCHEME: schedule

Normal Form:Third                         No. Decomps:     1
Lossless:    Yes                          Current No.:     1
Dependency Preserving:  Yes               No. Relations: 5

S H R                 Student
 ↑ Keys    FDs        Hour
 ↓ View               Room

R H T                 Room
 ↑ Keys    FDs        Hour
 ↓ View               Teacher

S C G                 Student
 ↑ Keys    FDs        Course
 ↓ View               Grade

R H C                 Room
 ↑ Keys    FDs        Hour
 ↓ View               Course

C T                   Course
 ↑ Keys    FDs        Teacher
 ↓ View

Display the FD's to this relation
```

S H R

Student	Hour	Room
Albert Harrington	4:00 Thu	3005 BSB
Albert Harrington	9:00 Mon	209 BH
Albert Harrington	10:00 Thu	209 BH
Anna Kammerling	10:00 Thu	100 SH
Anton Fabian	10:00 Thu	100 SH

R H T

Room	Hour	Teacher
3005 BSB	4:00 Thu	David Harris
3005 BSB	9:00 Mon	Tad Murata
209 BH	9:00 Mon	Edward Burack
100 SH	10:00 Thu	Edward Burack
100 SH	11:00 Wed	Edward Burack

S C G

Student	Course	Grade
Albert Harrington	EECS 345	D
Albert Harrington	EECS 202	A
Albert Harrington	EECS 499	A
Anna Kammerling	EECS 499	D-
Anton Fabian	EECS 499	D-

R H C

Room	Hour	Course
3005 BSB	4:00 Thu	EECS 345
3005 BSB	9:00 Mon	EECS 202
209 BH	9:00 Mon	EECS 499
100 SH	10:00 Thu	EECS 499
100 SH	11:00 Wed	EECS 499

C T

Course	Teacher
EECS 345	David Harris
EECS 202	David Harris

current catalog instance and can be subsequently examined and updated.

Any relation listed in a Decomposition Screen can be selected. This selection will have the effect of displaying a new Relation Screen containing the description of the relation. At this point, every menu available in the Relation Screen can be selected again. In particular, the designer can request to show an example table for the relation.

At present, the following algorithms support the functions provided in the Decomposition Screen:

- Decomposition into 3NF, dependency preserving, and lossless [Ul82]
- Decomposition into BCNF, lossless [TF82 and To87]
- Projection of a set of dependencies [Go87]

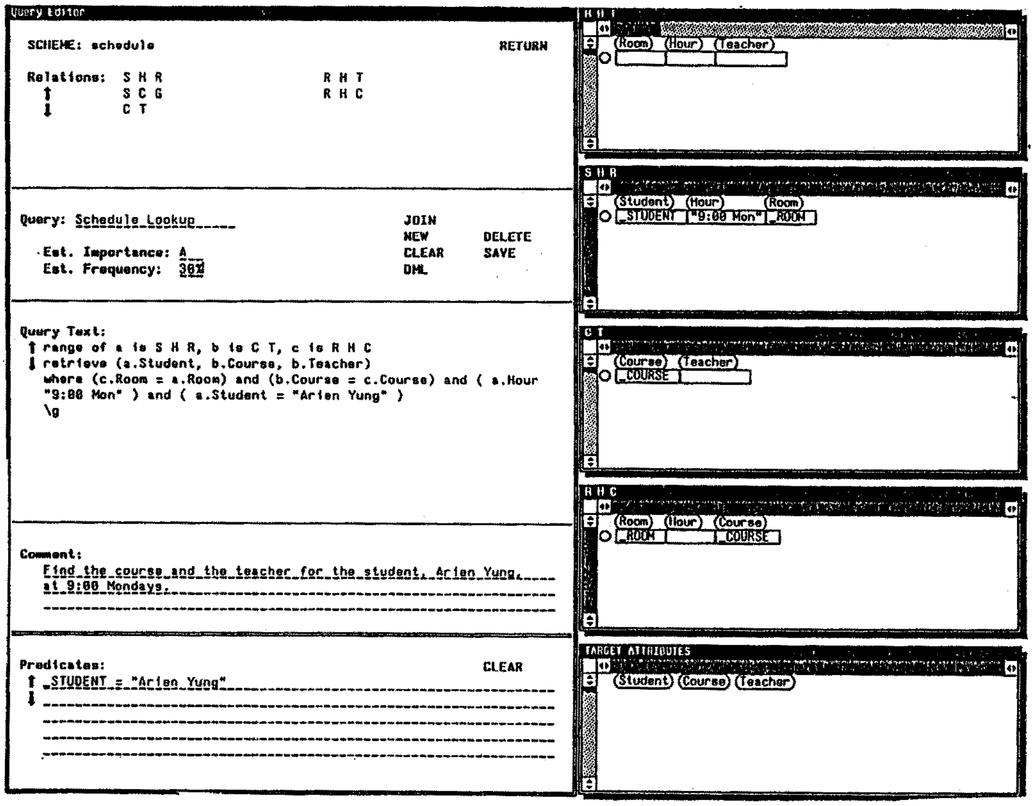
3. The Query Designer

A Query Design Screen (Figure 5) displays two representations of the query being designed. One is a graphical representation, similar to the one used in *Query-By-Example* [Zl77], where templates of the relations involved in the query are displayed. The other is text, in a conventional query language. In the current implementation, the query language is Quel since *DBE* interfaces with SunIngres. To design a query in *DBE*, the designer uses a screen editor and specifies the query graphically. When this specification is completed, the Query Designer automatically invokes an algorithm that finds the possible join paths and helps the designer disambiguate the query [To87]. Finally the *DBE* query is translated to its equivalent text query.

The graphical representation is more flexible, and does not require that the designer remember names of attributes or which relation an attribute belongs to. As a query is being designed, browsing through the catalog is possible and transition to another design screen, such as the Relation Screen is possible. Thus the designer can interleave the design of queries with the design of the database scheme.

The Query Screen is broken up into two parts (Figure 5). On the left hand side, the list of relations composing the current database scheme is displayed, and a number of subwindows contain commands to help specify and verify the query. The right hand side of the screen contains relation templates. A *DBE* query is constructed by

FIGURE 5 - QUERY DESIGN SCREEN



SCHEME: schedule

Relations: S H R, S C G, C T, R H T, R H C

Query: Schedule_Lookup

Est. Importance: A

Est. Frequency: 30%

Query Text:

```
range of a is S H R, b is C T, c is R H C
retrieve (a.Student, b.Course, b.Teacher)
where (c.Room = a.Room) and (b.Course = c.Course) and ( a.Hour
"9:00 Mon" ) and ( a.Student = "Arien Yung" )
\g
```

Comment:

Find the course and the teacher for the student, Arien Yung, at 9:00 Mondays.

Predicates:

```
_STUDENT = "Arien Yung"
```

R H T

Room	Hour	Teacher

S H R

Student	Hour	Room
_STUDENT	"9:00 Mon"	_ROOM

C T

Course	Teacher
_COURSE	

R H C

Room	Hour	Course
_ROOM		_COURSE

TARGET ATTRIBUTES

Student	Course	Teacher

selecting the relations participating in the query and filling out the relation templates. A number of editor commands and help menus are provided to assist the designer in this construction. Like in QBE [Zl77], a query can be specified by filling example elements and predicates in the attribute columns of the relation templates. Alternatively, predicates can be specified in the *Predicates* area at the bottom left of the query screen (Figure 5).

After the query is constructed and its syntax checked, it can be translated to text form by selecting the *DML* (Data Manipulation Language) menu.

One problem that we have begun investigating is how to verify the queries first against example tables, and later against an existing database. To solve this problem, *DBE* needs to interact with a database management system that contains a copy of the database against which the query will be run. This seems to be only possible for a database systems that can interpret queries on the fly. In the case of Ingres, it is possible to run queries interpretively, and so we can implement this feature.

Another advantage of using query translation is that it allows the design tool to have its own internal representation of the query that can be used with multiple database schemes. This feature adds flexibility in integrating the design of queries with the design of the database.

4. The DBE Design Catalog: Data Structures and Editors

The *DBE* design catalog stores all the design data and controls versions and history in the design process. It is structured to provide browsing and retrieval capabilities. The design catalog for a particular database contains a number of catalog instances. Domains are defined globally with respect to all catalog instances. All the other design entities (attributes, relations, tables, schemes, and queries) are defined with respect to one catalog instance.

A data structure and an associated set of functions are defined for each type of design entity. The functions typically include *create*, *view*, *copy* and *update*. They can be invoked by the designer through the corresponding editor (for instance, a new attribute can be created by invoking *the attribute editor*), or called by one of the design algorithms in the system. Below, we briefly describe these data structures and the specialized editors associated with them.

4.1. Catalog Instances

The catalog structure describes a version of the database design. Every catalog instance has a timestamp, indicating when it was created. When a new design for a database is initiated, a current catalog is loaded with default attribute domains (such as integer numbers and names). Then, as the design process proceeds, all attributes, dependencies, relations, and candidate schemes are specified with respect to the current catalog.

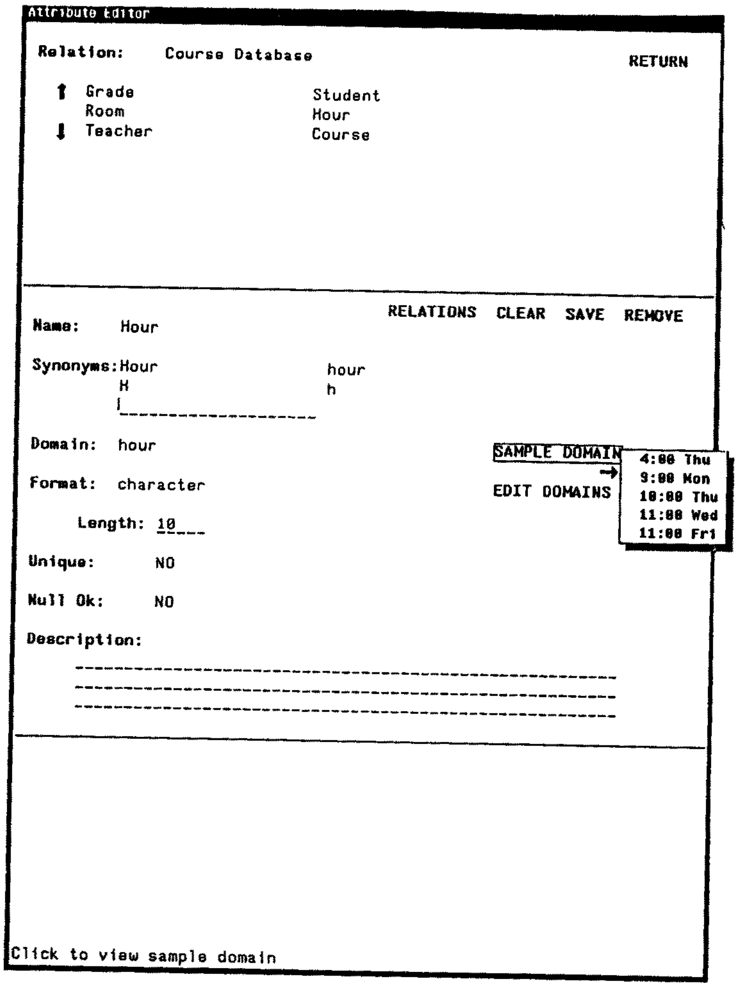
When a new catalog instance is created, information can be copied from other catalogs. Catalog instances are arranged in a tree hierarchy (Figure 1) according to the steps taken by the designer in the design process. Design entities can then be added to the new catalog, or entities copied from another catalog can be modified.

4.2. Attributes

The main components of the attribute structure are the attribute name and its domain. In addition, a data type and a format can be specified which define a storage class. For instance the basic type "integer" can support domains "employee-no" or "ssno", with different ranges and storage requirements. For every attribute, the designer can specify properties such as "unique", "null-allowed", a list of synonyms, and a short text describing its meaning.

Attributes can only be defined or updated using the *attribute editor* (Figure 6). This editor provides a main menu for specifying the attribute's properties. It also supports browsing capabilities on domain values, and other attributes or relations stored in

FIGURE 6 - ATTRIBUTE EDITOR



```
Attribute Editor
Relation:    Course Database                                RETURN

  ↑ Grade              Student
    Room               Hour
  ↓ Teacher            Course

                                      RELATIONS  CLEAR  SAVE  REMOVE
Name:     Hour

Synonyms:Hour                 hour
         H                    h
         |__________________

Domain:   hour                        SAMPLE DOMAIN  4:00 Thu
                                                  →  9:00 Mon
Format:   character                   EDIT DOMAINS   10:00 Thu
                                                     11:00 Wed
     Length: 10___                                   11:00 Fri

Unique:        NO

Null Ok:       NO

Description:
     ______________________________________________
     ______________________________________________
     ______________________________________________

Click to view sample domain
```

the catalog. The editor validates any input made by the designer, and maintains consistency in the dependencies and keys which the attribute participates in.

4.3. Domains

The domain structure specifies the range and type of values that an attribute may span. A domain has a name which is a unique identifier. There is a sample list associated with every domain, and functions are provided to add or delete values, sort the list, and test whether a value is included in it. The list can be explicitly specified by the designer and stored in the design catalog, or implicitly associated with a basic data type (e.g. integer). A number of default domains are predefined. Additional domains can be specified by the designer.

When defining a new attribute, the designer can browse through the list of values associated with any previously defined domain or request that a new domain be created. When an example table is generated by DBE or edited by the designer, a domain function is invoked that produces a stream of unique example values from that domain.

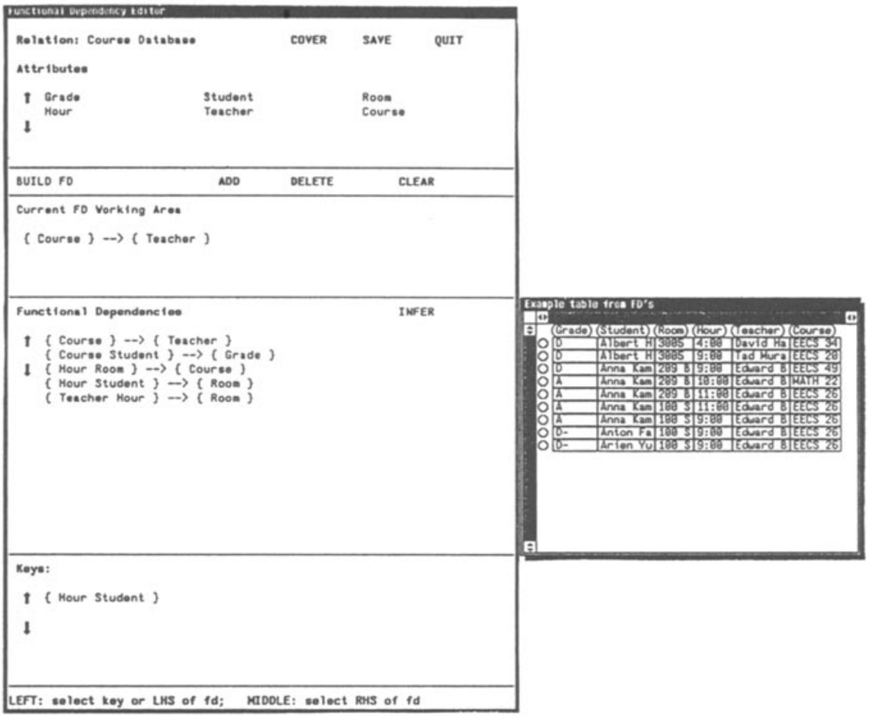
4.4. Dependencies and Keys

The dependency structure describes the type, functional or multivalued[2], and the left (lhs) and right (rhs) hand sides of the dependency. The lhs and rhs are pointers to lists of attributes that can have arbitrary length. Similarly keys are represented as pointers to a list of attributes.

A *dependency editor* (Figure 7) can be invoked to specify new dependencies or update previously defined dependencies. A number of specialized features are associated with the editor. The editor will check consistency of any new dependency before adding it to the current set. Keys are dynamically found and displayed, and the set of dependencies can be reduced to a more compact cover by selecting the *COVER* menu. Two covers are generated: a *minimal cover* and a *canonical cover* [Ul82]. The

[2] At present, multivalued dependencies are only partially implemented in *DBE*. They will be more systematically integrated in the system at a future stage.

FIGURE 7 - FUNCTIONAL DEPENDENCY EDITOR



```
Functional Dependency Editor

Relation: Course Database                 COVER      SAVE      QUIT

Attributes

↑  Grade                 Student                 Room
   Hour                  Teacher                 Course
↓

BUILD FD                    ADD         DELETE          CLEAR

Current FD Working Area

 { Course } --> { Teacher }

Functional Dependencies                                 INFER

↑  { Course } --> { Teacher }
   { Course Student } --> { Grade }
↓  { Hour Room } --> { Course }
   { Hour Student } --> { Room }
   { Teacher Hour } --> { Room }

Keys:

↑  { Hour Student }

↓

LEFT: select key or LHS of fd;    MIDDLE: select RHS of fd
```

Example table from FD's

Grade	Student	Room	Hour	Teacher	Course
D	Albert H	3005	4:00	David Ha	EECS 34
D	Albert H	3005	9:00	Tad Mura	EECS 20
D	Anna Kam	209 B	9:00	Edward B	EECS 49
A	Anna Kam	209 B	10:00	Edward B	MATH 22
A	Anna Kam	209 B	11:00	Edward B	EECS 26
A	Anna Kam	100 S	11:00	Edward B	EECS 26
A	Anna Kam	100 S	9:00	Edward B	EECS 26
D-	Anton Fa	100 S	9:00	Edward B	EECS 26
D-	Arien Yu	100 S	9:00	Edward B	EECS 26

designer can choose to replace the set of dependencies by one of them.

The screen displayed by the functional dependency editor displays the attributes in the relation, and the list of functional dependencies and keys currently defined. A work area is provided (*Current FD Working Area*), where a new key or a new dependency can be constructed by selecting attributes with the mouse. As dependencies are added, the system automatically finds the keys and displays them. Likewise, when a key is specified by the designer, the corresponding functional dependency is added to the current set. The editor only saves a unique instance of a dependency. It also provides a menu selection to reduce a partially constructed set of dependencies to a minimal or canonical cover [Ul82].

4.5. Relations

The relation structure has a unique name; a set of attributes; a set of functional and multivalued dependencies; tags indicating its degree of normalization (3NF, BCNF, 4NF); and an associated instance, the *example table*. Additional properties in the relation structure link a relation to a database scheme which contains this and other relations.

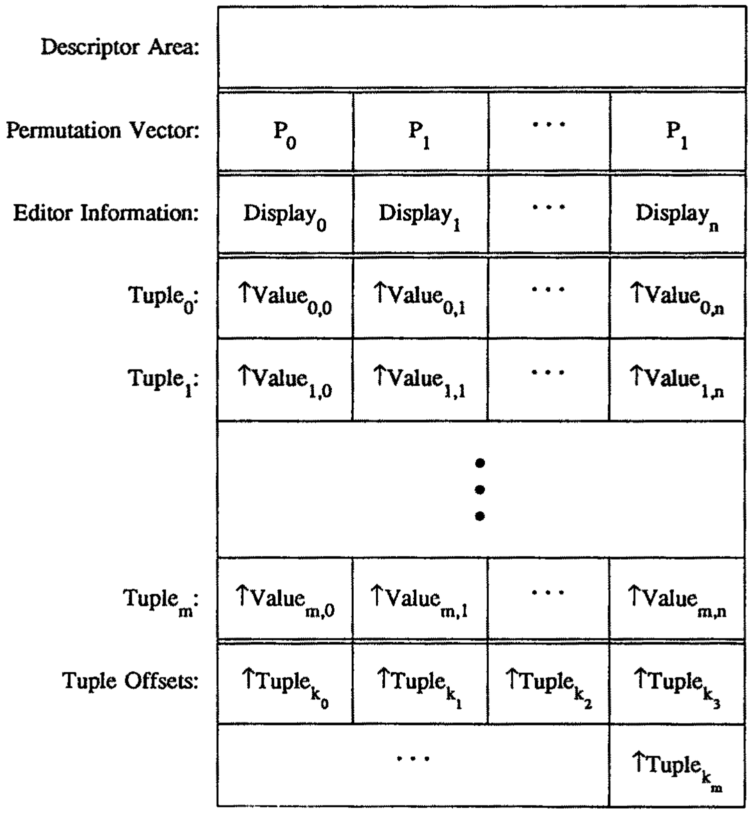
The relation data structure constitutes the core of the design tool. Most of the design algorithms used by *DBE* are invoked in the context of specifying or updating a relation description. The designer specifies a relation by using the attribute, dependency, and table editors. Consistency within the design catalog is automatically enforced by the editors.

4.6. Tables

Any relation stored in the catalog may have an associated instance that contains data values sampled from actual attribute domains. This instance is an *example table*, and it is extensively used to assist the designer in specifying the semantics of the database. The designer can load an example table containing sample data and ask the system to infer dependencies from it, or dependencies can be specified directly, using the dependency editor, and the system will generate an example table which is an exact representation of the dependencies. In this case, the example is an Armstrong Table [Ar74, BDFS84].

The table structure is a space-efficient data structure that allows storage, addition, and deletion of tuples (Figure 8).

FIGURE 8 - DBE TABLE STRUCTURE


Descriptor Area:
Permutation Vector:
P_0
P_1
$\cdots$
P_1
Editor Information:
Display_0
Display_1
$\cdots$
Display_n
Tuple_0:
$\uparrow\text{Value}_{0,0}$
$\uparrow\text{Value}_{0,1}$
$\cdots$
$\uparrow\text{Value}_{0,n}$
Tuple_1:
$\uparrow\text{Value}_{1,0}$
$\uparrow\text{Value}_{1,1}$
$\cdots$
$\uparrow\text{Value}_{1,n}$
Tuple_m:
$\uparrow\text{Value}_{m,0}$
$\uparrow\text{Value}_{m,1}$
$\cdots$
$\uparrow\text{Value}_{m,n}$
Tuple Offsets:
$\uparrow\text{Tuple}_{k_0}$
$\uparrow\text{Tuple}_{k_1}$
$\uparrow\text{Tuple}_{k_2}$
$\uparrow\text{Tuple}_{k_3}$
$\cdots$
$\uparrow\text{Tuple}_{k_m}$

At the top of the table, a table descriptor is stored, followed by an array of tuples. Tuples are implemented as arrays of pointers to character strings. At the bottom of the table, tuple offsets are stored, with a free area left in the middle of the table for inserting new tuples. The table is automatically expanded when this free area is exhausted.

The table descriptor contains the number of tuples stored and the maximum number of tuple slots allocated. It also contains information required by the table editor such as a permutation vector indicating the order and format in which the attributes are displayed by the editor. This vector is updated when the user requests viewing the columns in a different order. Similarly, if the user wishes to sort or modify the tuples in some way, this transformation can be done by changing only the array of tuple offsets.

The *table editor* displays a 2-dimensional table representing an instance of a stored relation scheme. Horizontal and vertical scrolling are available to facilitate examining a table which may not fit in the editor window. The table editor supports a range of functions for manipulating columns (attributes), rows (tuples), and fields. These functions appear as pop-up menus when the appropriate column, row, or field is selected. Columns can be permuted, hidden, or reformatted to a desirable width. The values in a column can be sorted or randomized. Rows can be added, deleted, or copied. Field values can be modified. All updates are instantaneously reflected in the window display.

4.7. Schemes

A scheme structure has a unique name, and an associated set of relations. It also includes tags indicating whether the scheme is normalized (3NF, BCNF, 4NF), lossless, and dependency preserving. A scheme can be constructed by the designer, or generated by decomposing a relation into a normal form.

To define a scheme, a screen called the *Scheme Viewer* is provided (Figure 2) which provides menu selections to incrementally add relations and attributes to the scheme. As with the attribute editor, consistency with the *DBE* design catalog is checked before any input is accepted.

5. The DBE Storage Subsystem

In order to provide flexibility in the design process and specialized assistance in the specification of a database and its queries, *DBE* must store and manipulate a considerable amount of design data. Furthermore, the design algorithms utilized produce large sets of intermediate results. For instance, for every pair of tuples in a table, the dependency inference algorithm constructs a number of subsets of attributes that grows exponentially with the number of attributes in the table. Thus it is important to store these subsets as compactly as possible.

Relational design algorithms operate on sets of attributes, representing relations or left and right hand sides of logical dependencies. Thus it is important to efficiently support set operations such as union, intersection, subset-of, etc. In *DBE*, attribute sets are internally represented as binary vectors. The attributes in a database are ordered, $R = A_0, A_1, \cdots, A_{n-1}$. A set $S \subseteq R$ is represented as a binary number $(i_0\, i_1 \cdots i_{n-1})$, with $i_j=1$ if A_j is in S, and $i_j=0$ otherwise. For instance, in the database *R=Course Teacher Student Grade Hour Room* , the set *Course Student* is represented by the binary number (101000).

With this data structure, we are able to support an efficient implementation of set operations (union, intersection, difference), boolean operations (set-membership, subset, proper subset), and functions to add or remove an attribute from a set. For most pratical databases, these operations are accomplished in a few machine instructions. The set functions constitute a basic layer of *DBE*, and are invoked by the higher level functions.

Storage and manipulation of character strings are optimized by keeping all strings in one system table. *DBE* maintains the invariant that all strings are stored only once. Strings are stripped of leading and trailing blanks and inserted into a hash table. A search or insertion return a pointer to the unique instance. String comparison reduces to testing pointer equality.

Example tables are stored in a compact structure (Figure 8), containing descriptors at its top and tuple pointers at its bottom. All example tables used during a design session are allocated memory within one large static area. Within this area, they can be relocated when space is needed for other example tables. Tuple offsets, stored at the bottom of the table are relative to the top of the table.

Finally, *DBE* maintains its own free lists and heaps of memory, instead of allocating memory globally by making an operating system call (the *sbrk()* Unix call). Structures such as sets and dependencies are kept on separate free lists and are allocated and freed on a per structure basis. In addition to being space efficient, this memory management scheme may cut down on the amount of paging for large database design.

6. Summary and Future Work

DBE is a tool for designing relational databases and queries. One of its distinguishing features is that it provides expert assistance in specifying attributes and logical dependencies, without departing from the relational model. This assistance is provided in the form of graphically displayed examples and editors that interface with an intelligent design catalog. In particular, the designer has the option of examining and modifying example tables as a way to verify the specification of logical dependencies.

Another important feature of *DBE* is that it integrates the design of queries with the design of a database scheme. The designer is given the option to consider alternative database schemes and experiment with formulating relational queries against them. Like in the specification of the logical dependencies, *DBE* assists the designer in disambiguating a query by generating dual representations (graphical and text) and using examples.

Consistency of the design catalog is automatically maintained by minimizing redundancy and concurrently updating all related design entities and all their representations. For instance, the example table and the set of functional dependencies associated with a relation are always consistent with each other, as are the two representations of a query.

In the implementation of *DBE*, we have emphasized performance issues. A customized storage subsystem provides a compact organization of the design information and supports an efficient implementation of the design algorithms. Currently, the entire *DBE* system consists of 45,000 lines of C code. The prototype runs comfortably with 4 megabytes of main memory on a Sun 3/50 (a 1 MIP machine).

We have recently performed an extensive set of experiments that demonstrate the feasibility of our approach to automatic database design. For instance, we have showed that our implementation of the dependency inference function leads to acceptable interactive response times for realistic example tables [BMOT87].

In the future, we plan on expanding *DBE* in two main directions. One is adding functionality to the Query Designer and investigating better techniques for integrating it with the Schema Designer. We are currently working on a characterization of join paths, and investigating types of updates to a database scheme that could resolve ambiguities in a relational query [To87]. The other natural expansion would be to add a physical design component to the system. This component could be a front-end to the query optimizer of the database system for which *DBE* generates a schema and queries. Finally, an open question is whether it would be possible to develop a higher-level (non-relational) design interface on the top of *DBE*, while preserving the major features of the system.

Acknowledgements

Work on the *DBE* system was initiated in collaboration with Kari-Jouko Raiha at Cornell University [BMR85]. The current prototype was developed at the University of Illinois at Chicago. We thank Soon-Ho Jeon and Arien Yung for their contributions and long hours of programming in the final stages of implementation. Arien wrote the screen interpreter and the functional dependency editor. Soon-Ho helped design and implement the driver and the history mechanism.

References

[Ar74] Armstrong W.W., "Dependency Structures of Data Base Relationships," *Information Processing* , 74, 1974. Berlin, August 1978.

[BDFS84] Beeri C., Dowd M., Fagin R., and Statman R., "On the Structure of Armstrong Relations for Functional Dependencies," *Journal of the ACM*, 31:1, 1984.

[BMR85] Bitton D., Mannila H., and Raiha K., "Design-By-Example, A Design Tool for Relational Databases," *Technical Report*, Cornell Univ., March 1985.

[BMOT87] Bitton D., Millman J.C., Orji C., and Torgersen S., "A Feasibility and Performance Study of Dependency Inference," *Technical Report* , University of Illinois at Chicago, Dec. 1987.

[F82] Fagin R., "Armstrong Databases," *IBM Research Report RJ3440*, San Jose, California, 1982.

[Go87] Gottlob G., "Computing Covers for Embedded Functional Dependencies," *Proceedings ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, 1987.

[LO78] Lucchesi C.L. and Osborn S.L., "Candidate keys for relations," *Journal of Computer and System Sciences*, 17:2,1978

[MU83] Maier D. and Ullman J.D., "Maximal Objects and the Semantics of Universal Relation Databases," *ACM Transactions on Database Systems*, 8:1,1983.

[MMS79] Maier D., Mendelzon A.O., and Sagiv Y., "Testing Implication of Data Dependencies," *ACM Transactions on Database Systems*, 4:4,1979.

[MUV84] Maier D., Ullman J.D., and Vardi M.Y., "On the Foundations of the Universal Relation," *ACM Transactions on Database Systems*, 9:2,1984.

[MR86] Mannila H. and Raiha K.-J., "Design by Example: An Application of Armstrong Relations," *Journal of Computer and System Sciences*, 33, 2, Oct 1986.

[MR87] Mannila H. and Raiha K.-J., "Dependency Inference," *Proceedings Thirteenth International Conf. on Very Large Data Bases*, Brighton, August 1987.

[MZ80] Melkanoff M.A. and Zaniolo C., "Decomposition of Relational and Synthesis of Entity-Relationship Diagrams," *Entity_relationship Approach to System Analysis and Design*,(P.P. Chen ed.),North-Holland Publishing Company, 1980.

[SM81] Silva A.M. and Melkanoff M.A., "A method for helping discover the dependencies of a relation," *Advances in Data Base Theory*, 1, Gallaire H., Minker J., Nicolas J.M., Plenum Press, 1981.

[To87] Torgersen S., *Ph.D. Thesis Proposal*, Cornell University, Dec. 1987.

[TF82] Tsou D.M. and Fischer P.C., "Decomposition of a Relation Scheme into Boyce-Codd Normal Form," *Proceedings ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, 1982.

[Ul82] Ullman J.D., *Principles of Database Systems (Second Edition)*, Computer Science Press, Rockville, Md., 1982.

[Zl77] "Query-By-Example: A Database Language," *IBM Systems Journal*, 16:4, 1977.