Computer Aided Database Design

Karin Beck, Korbinian Kern, Lore M. Kern-Bausch, Bernd G. Wenzel

GEMAP mbH, Munich

Abstract

For all DBMS-based software systems, a good database design is the key to performance. Nevertheless the necessity to support this important task by adequate methods and comfortable tools has been neglected up to now. As a consequence at least one generation of computer programmers fought the battle for necessary performance which was already lost in the beginning, lost by design.

Facing this situation, GEMAP first developed a database design methodology called Information Analysis. As database applications become more and more common and as they show increasing complexity today, GEMAP started the development of INFODIC, the GEMAP INFOrmation DICtionary, a tool for automatic database design based on Information Analysis.

This paper will present to you an overview of this development. The first section outlines the basic concepts of our methodology, Information Analysis. The second section shows how the interactive components of INFODIC support the information analyst in entering and maintaining the information structure. The third section covers the topic of consistency checks during all phases of Information Analysis. The fourth section gives you an impression of the resulting table and index structures for a relational database. The final one contains an outlook on our plans, coming enhancements and future work.

Keywords

CASE Tool, Semantic Data Modell, Information Analysis, Conceptual Schema, Data Dictionary, Database Design, Tables and Indexes.

Contents

Introduction

INFODIC, the GEMAP-INFOrmation-DICtionary, is a design tool for conceptual database modelling.

Based on a semantic datamodel, the information analysis [7], INFODIC describes the information structure of an application and automatically derives the relational database schema.

The principle of INFODIC is the object role modelling method, which is based on the semantics of the natural language ([6], [5], [8]).

1 Information Analysis

The aim of information analysis is to set up the conceptual schema, that means the description of the information structure of an application, which is not subject to change as the application evolves. The structural concepts, used for this purpose, are objecttypes, associationtypes and dependencetypes.

1.1 Objecttypes

An objecttype is a class of things (real or abstract) which is interesting in the context of the application area. All the different informations in a database are informations about objects of some type.

For the following, a bill of material problem should serve as an example: Information concerning parts contained in components (with quantity QTY) and their deposits, as well as information about part suppliers, their address and phonenumber.

Using the graphical notation of figure 1 for the results of information analysis, objecttypes are represented by rectangles. So e.g. "part" and "company" are objecttypes.

1.2 Semantically Irreducible Associationtypes

An associationtype is a class of informations about objects of given types. It can be considered as a pattern of a natural language sentence.

To avoid instabilities in the information structure, the associationtypes have to be semantically irreducible. This means that it must not be possible to decompose an associationtype into one or more others without loss of information.

In our graphical notation the circular constructs represent associationtypes. The associationtype "part_suppl" e.g. between the objecttypes "part" and "company" represents that parts are supplied by companies. The objecttype "company" plays the role "supplies" in this associationtype, the objecttype "part" that of "supplied_by".

A role is connected to exactly one objecttype, which means that only one objecttype can play a special role in an associationtype.

Along this connection line you read the lower and upper bounds of the corresponding membership interval ("*" means unlimited). It has to be read from the objecttype (rectangle) to the role. The lower bound tells us, in how many occurrences of the associationtype any occurrence of the corresponding objecttype at least has to be found playing this role. The upper bound tells us the maximum.

So in the example, a "part" is supplied by exactly one "company", but one "company" may supply any number of "parts" (including 0). The lower bound of 0 means that we are interested also in companies which do not supply any parts for the moment.

It is important to note, that an associationtype itself may be used as an objecttype in another associationtype, as the example of figure 1 shows. Here the associationtype "BOM_entry" plays the role "has QTY" in the associationtype "BOM_QTY".

1.3 Dependencetypes

Dependencetypes are informationtypes of higher order (correlations between informations). They look formally like associationtypes.

They are especially of interest as integrity assertions for databases and as rules for expert systems.

INFODIC is able to record dependencetypes, but they are not yet taken into consideration in a database design. Therefore we don't want to cover these topics here and the example doesn't contain any dependencetypes.

1.4 The Phases of Information Analysis

It proved very useful to distinguish the following phases in modelling reality using information analysis.

First, we ask for all the objecttypes we see immediately in our area of interest, we collect all the information (associationtypes) relevant for our objecttypes, and we document hierarchical relationship (subtypes) between our objecttypes.

The second step is iterated for every objecttype in the model until no more new objecttypes appear. Here we ask for the naming conventions for the objecttypes. It has to be stated here that a nametype is an objecttype which is used to reference another objecttype (uniquely or not), that a naming convention is a special case of an associationtype, which associates an objecttype with its nametype.

Figure 2 shows two naming conventions, one for "part" which can be referenced uniquely by "part_no", and one for "company" which can be referenced also uniquely by "name".

In addition, we ask for the attributes (interesting information) of our objecttypes and keep record of the objecttype hierarchies. We also fix the physical representation of the objecttypes.

The dotted arrows in figure 2 are the graphical notation for the physical representation of objecttypes.

1.5 Deriving Relations and Access Paths

In order to derive table structures, we build the connectivity components of those connections in our graphical representation which are labeled with a membership interval of 1:1 only, if we want to end up with 5NF relations, or with a membership interval of 0:1 in addition, which leads - with respect to performance - to an optimal design for an RDBMS (Relational DataBase Management System) handling null values correctly and efficiently (e.g. ORACLE).

Figure 3 shows the connectivity components for the bill of material problem, also the 0:1 intervals are taken into consideration.

Every such component results in one table, which contains one column for any objecttype inside the component and one column for every role, which leads out of the component. All nonlexical objecttypes have to be replaced here by an unique name (naming conventions).

Every objecttype inside of a connectivity component, as mentioned above, is a candidate key. For every candidate key a unique access path should be implemented in order to support database integrity. Also performance is enhanced by such an access path.

Additionally we see foreign keys, this means all the roles a candidate key plays in other components. For these foreign keys additional access paths are to be provided. In database systems access paths are usually supported by indexes.

The result of the information analysis for the bill of material problem is shown in figure 4: The relation schema, the unique indexes (underlined column names) and the foreign key indexes (join columns indicated by the arrows).

2. INFODIC as an Information Analysis Tool

Beyond question, information analysis is an exact method, but the result is a very expensive one, because there are lots of details to document. Of course, large systems can be maintained with editors, but a special tool, checking the consistency and taking care of the context, would offer great support.

INFODIC is such a tool. It allows the user to view an information structure, resp. a meaningful part of it, to extend and modify it while a maximum of consistency is guaranteed.

INFODIC is fully database supported (for the time being: ORACLE). The architecture of INFODIC consists of two components, an interactive one to record and maintain information in order to keep it consistent, and a component to check the information analysis and derive the relational database design, using a precompiler as interface to the INFODIC database.

The interactive part offers masks which a user can fill with the relevant information. All masks are built in such a way that the user is guided from introducing fundamental input up to the completion of the infomation analysis. All the time it is possible to delete or extend, and the input is checked for consistency automatically. All masks are formed with SQL*FORMS V 2.0 ORACLE V 5.1.

2.1 Entering the Relevant Information

A basic mask serves for entering and showing the basic information relevant for the application area. First, all elements can be filled in without being fully specified. Elements are objecttypes, associationtypes and dependencetypes. If the user already knows the refined structure of his element, he can write it also in the basic mask, e.g. the declaration if an objecttype is a lexical or a nonlexical one.

Figure 5 shows this mask with parts of the relevant information of our bill of material problem (see figure 1 also).

A particular mask serves for long descriptions and comments.

2.2 Entering and Maintaining Objecttypes

The mask in Figure 6 describes the refined structure of objecttypes only. Here the user must specify things like the physical representation of a lexical objecttype by a basetype (BOOL, STRING, INT,...) and its length.

Declarations in connection with special objecttypes, like set or enumeration types, are to be fixed in an additional mask.

All dates concerning names and types are mandatory, other dates are typedependent and they will be checked against consistency later.

The possibility to delete separate objecttypes, which are not involved in any associationtype, is also part of the maintenance. INFODIC checks such wishes and denies deletion, whenever an objecttype belongs to any associationtype.

2.3 Entering and Maintaining Associationtypes and Dependencetypes

For these purposes, there exists a mask, too. For the bill of material problem, figure 7 shows the input possibilities of associationtypes resp. dependencetypes.

Beneath the title "roles", the associations and dependences will be defined. For this purpose, the associationtypes are specified with the involved objecttypes, their roles and membership intervals.

INFODIC guarantees that associations, for which the objecttypes participated in it have already been inserted, can only be renamed but not deleted. Furthermore it is guaranteed that role names and membership intervals, with lower bounds not greater than the upper ones, have to be inserted.

As an example for naming conventions, the identification for the nonlexical objecttype "part" is given in figure 8. INFODIC prepares the name for such a naming convention in figure 8: "NAM-CON-2". The user has to fill in the objecttype being identified (part), the identifying objecttype (part_no) and the concerning membership intervals.

3 Consistency Checks in INFODIC

INFODIC grants the consistency of the information by a lot of checks during and after entering the information.

As indicated in Section 2, the first checks are done during information analysis. Further checks cannot be done before the inserting is completed. Concluding checks help to prepare the derivation of the relational schema. The two latter ones are mainly implemented with ORACLE-precompiler calls embedded in C-programs.

3.1 Consistency for the Information Structure

To guarantee the consistency of the whole information structure, information analysis can't be closed without checking that the information is complete, correct and without contradiction.

Among others the following rules are checked:

- Representation rules for lexical objecttypes
  - . basetypes (BOOL, DATE, STRING, INT) must not be represented
  - . any other lexical objecttype must be represented
  - . exception: a subtype of an objecttype fitting one of the two forstanding items does not need it's own representation

- Rules for nonlexical objecttypes

  . any nonlexical objecttype must have an unique naming convention

- Rules for hierarchical relations between elements

  . element A is not allowed to be subtype of itself

  . if element B is subtype of element A, A cannot be subtype of B

  . if element C is subtype of element B and B is subtype of A, C is also subtype of A

3.2 Consistency Checks in Preparing the Database Design

Especially the membership intervals allow INFODIC to check some semantic rules, which are fundamental to derive a database free of redundance.

For instance the following checks are implemented

- reducible associationtypes are recognized and notified as an error
- the existence of undefined occurrences of an objecttype (only zero lower bounds in all concerning membership intervals) are notified as a warning
- nonlexical objecttypes not involved in any association are not considered in the database design.

4 INFODIC as a Design Tool for Relational Databases

So far, INFODIC has been presented as a very general tool for the formal modelling of all kinds of information structure. Neither the basic method nor the tool with its logic concerning the recording and the consistency rules does really restrict the user in any case. Therefore it is not astonishing that this method is very suitable for fitting formally and generally into the rules of the relational database model ([2], [3], [4]).

4.1 Generating Tables

A relational database consists of tables (relations). These are sets of identically structured data records with elementary data fields.

In nearly all cases, INFODIC is able to derive an optimal table structure out of the existing information structure. For DBMS (DataBase Management System), based on SQL, the tables are built with the standard SQL-statement CREATE TABLE, e.g.:

```
CREATE TABLE company
            (name            CHAR (40)     NOT NULL,
            is_phone_of      CHAR (20)     NOT NULL,
            is_address_of    CHAR (120)    NOT NULL)
```

The necessary datatype adaption, dependant on the target system is INFODIC's job. For other than SQL based database systems, it's not a principal problem to generate table descriptions with another syntax like QUEL, for instance, instead of SQL (for INGRES). If required, the implementation will only need a few days.

The reason for the remark, concerning a restricted optimality, lies in the present release of INFODIC, which doesn't take into account the influence of dependences onto the table structures. But as we know from experience, only one percent of the tables created by INFODIC show an insignificant loss of optimality. Working with such a database, however, is without restrictions always possible.

E.g.: Very seldom and only in very complex database design problems, a data field may exist twice. In any case, the result with INFODIC will be better than the solution of the most experts, and by far better than any ad hoc solution, done by feeling without a method.

INFODIC's possibilities are widely spread. On the one hand, the tool supports bill-of-material-like structures, as we have seen; not all database design tools are able to solve such problems (e.g. [1], [9]). On the other hand INFODIC even generates an optimal database structure, when other methods like the normalization theory must give up. This happens mostly in case of intended redundancy, e.g. striking the balance is principally always possible, when the relevant data can be accessed, and it is not necessary to store it, but storing the balance can lead to a better performance. INFODIC does allow to store redundant information.

Summarizing, INFODIC applies an optimal table schema for a relational database nearly completely and automatically. If this is not possible, it supports such a table schema at least very efficiently.

4.2 Generating Indexes

INFODIC however not only supports the development of an optimal relation schema, it also supports the development of optimal access paths on a relational database. It is important here to mention, that the separation of data structures (tables) and access paths (indexes), usually seen in the relational data model, must be done strictly, because it is important for developing secure application software; but access paths are also absolutely necessary for tuning aspects, in order to allow an efficient and therefore also economic use of a database.

INFODIC proposes a set of indexes derived from the existing information structure. For DBMS based on SQL, indexes are built with the standard SQL-Statement CREATE INDEX, e.g.:

```
CREATE UNIQUE INDEX company_name ON company (name)

CREATE INDEX part_supplied_by ON part (supplied_by)
```

The necessary adaptions to the target system are done by INFODIC. Also, in this case it's no principal problem to generate index definitions with another syntax. When required, the implementation will only need a few days.

The proposed index set contains unique indexes as well as all indexes, necessary for reasonable joining different tables. Especially all indexes which can guarantee the consistency of the database belong to this set. These indexes support efficiency, too. Extreme requests, concerning performance and/or the amount of data may require special tuning actions.

Performance aspects could cause the need to changing the sequence of the data fields in a concatenated index. E.g. the index COMPANY_NAME supports all the accesses onto the companies' name. If lists, sorted via phone numbers, are a main application, it would be better to have an index COMP_PHONE with the field sequence phone and name.

```
CREATE UNIQUE INDEX comp_phone ON company (phone, name)
```

Moreover, dependent on the target system, it might be useful to generate not all indexes proposed by INFODIC. So relational database systems may have problems with index fields not occupied. The proposed index list however is of essential help for the database administrator and it is very easy to select those indexes, supported by the target system.

Besides, it is not possible to extract all desired accesses out of the information structure. E.g. attributes serving as database entries often depend on application details. These indexes must be added. This should be no problem, because the user knows the corresponding data fields and he can induce INFODIC to create the index.

It should be mentioned, that the INFODIC index proposal is to be supplemented by different declarations depending on the target system, e.g. the index block size, the index organization as B-tree resp. hashing and so on. But those declarations are beyond the intended aims of INFODIC at the time.

Summarizing it can be stated, that the set of indexes proposed by INFODIC is a first but good approach. It contains all indexes to guarantee structural consistency, also those eventually unknown to the user. Therefore INFODIC simplifies the tuning of a database immensely.

4.3 Support of Other Data Models

Though the title of section 4 is restricted to relational databases, it should be mentioned that INFODIC doesn't favour relational database systems. Table structures generated by INFODIC can be used as recordtypes for every network or hierarchical database. Especially because INFODIC is able to propose hierarchical structured records, typical for classic database systems as well. These recordtype definitions have to be supplemented by the unique identifiers. This is no problem because the corresponding information is available (as already mentioned, INFODIC can create unique indexes).

Principally, the relations between recordtypes are also completely available in INFODIC. Therefore the transformation to set definitions in a network schema can be done without difficulties. In case of a hierarchical database system, those relations directly supported by the database software must be added. But this step requests a detailed knowledge of all applications, because the performance is thereby influenced in a high degree.

5 INFODIC as a Programming Tool

INFODIC - and moreover the method of information analysis it is based on - does not only support database design. The application facilities are multivarious ones. But the actual version of INFODIC does not yet cover all principal possibilities.

5.1 Generation of Data Structures

Yet INFODIC generates record structures as INCLUDES corresponding to the database records, which the programmer can copy in COBOL- PASCAL- and PL/I programs. Thus the standard problem of these languages - losing the hierarchical record structure when facing relational databases - is avoided.

5.2 Outlook

In the future, there is a new kind of software development growing up. It should be possible to generate applications nearly automatically, based on the knowledge of the input parameter, the necessary information and the general information structure. Then the programmer only will have to show the system on the map of the information structure the starting point, the intermediate stations and the target information.

The actual status of INFODIC is a software system able to generate databases, and therefore it supports an area of software development, whose economic meaning is often underestimated.

But in future INFODIC will hopefully mature to a very general software development tool.

References

[1] R. Barker
SQL*Design Dictionary (SDD)
Proc. 5th European Oracle Users Group Conference,
Munich, April 1987

[2] E. F. Codd
A Relational Model for Large Shared Data Banks
CACM, Vol. 13, No.6,
Juni 1970

[3] E. F. Codd
Further Normalization of the Database Relational Model
in R. Rustin (Ed.)
Data Base Systems Courant Computer Science Symp., Vol.6 Prentice Hall,
Englewood Cliffs,
N.J., 1972

[4] R. Fagin
Multivalued Dependences and a New Normal Form for Relational Databases
ACM Transactions on Database Systems
Vol. 2, No3,
September 1977

[5] E. Falkenberg
Concepts for Modelling Information
in G. M. Nijssen (Ed.)
Modelling in Database Management Systems,
North-Holland, 1976

[6] W. Kent
Data and Reality
North-Holland, 1978

[7] L. M. Kern-Bausch, B. G. Wenzel
Database Design for Relational Systems: Why, Who, How?
J. Timm, Ph. Lord (Ed.)
European ORACLE Users Group Newsletter,
No10,
August 1986

[8] G. M. Nijssen
A Gross Architecture for the Next Generation Data Base Management Systems
in G. M. Nijssen (Ed.)
Modelling the Data Base Management Systems,
North-Holland, 1976

[9] Quint
TINA Reference Manual
Quint Database Systems Corporation,
July 1985

Bill of Material Conceptual Schema

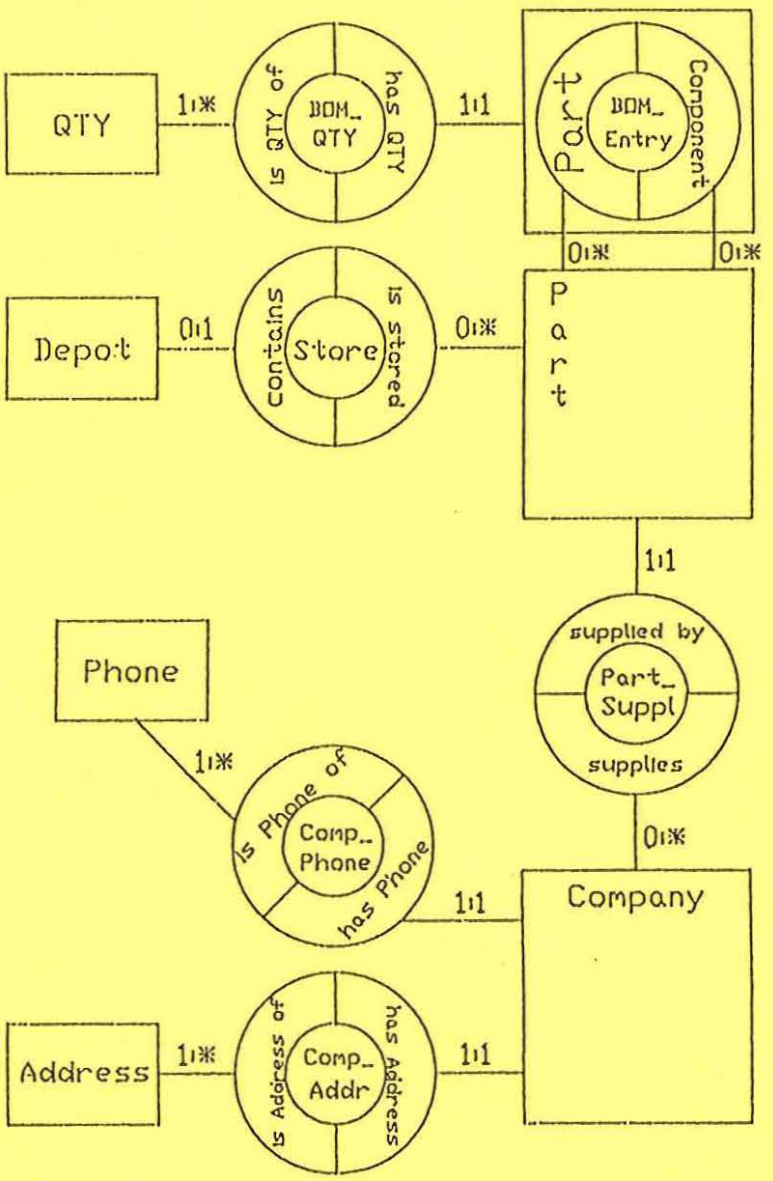

Figure 1

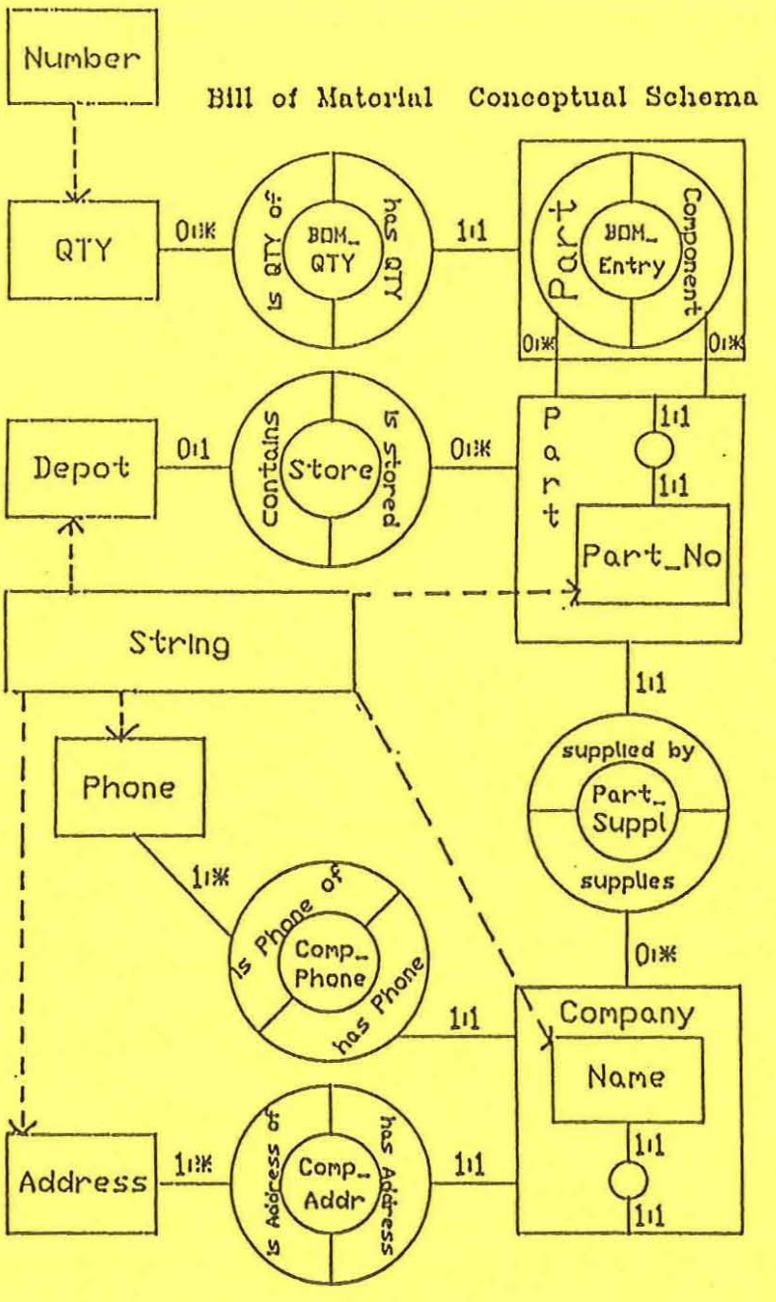
Number
Bill of Material Conceptual Schema
QTY
0:K
is QTY of
BOM_ QTY
has QTY
1:1
Part
BOM_ Entry
Component
0:K
0:K
Depot
0:1
contains
Store
is stored
0:K
Part
1:1
1:1
Part_No
String
1:1
Phone
supplied by
Part_ Suppl
supplies
1:K
is Phone of
Comp_ Phone
has Phone
0:K
1:1
Company
Name
Address
1:K
is Address of
Comp_ Addr
has Address
1:1
1:1
1:1
© 1986 GEHAP mbH

Figure 2

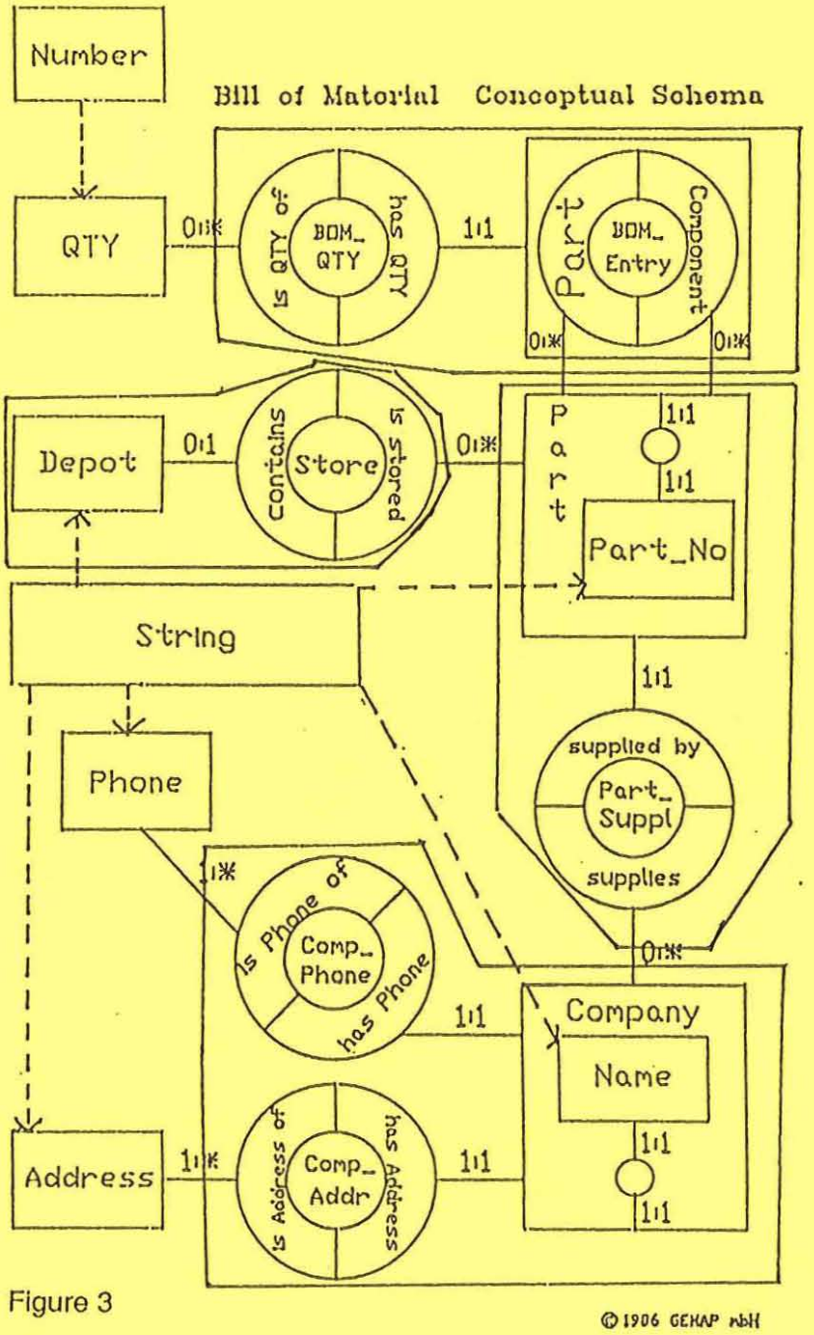
Number
Bill of Material Conceptual Schema
QTY
0:*
Is QTY of
BOM_ QTY
has QTY
1:1
Part
BOM_ Entry
Component
0:*
0:*
Depot
0:1
contains
Store
is stored
0:*
Part
1:1
1:1
Part_No
String
1:1
Phone
supplied by
Part_ Suppl
supplies
1:*
Is Phone of
Comp_ Phone
has Phone
0:*
1:1
Company
Name
Address
1:*
Is Address of
Comp_ Addr
has Address
1:1
1:1
1:1
© 1986 GEHAP mbH

Figure 3

Relation Schema

+

Indexes

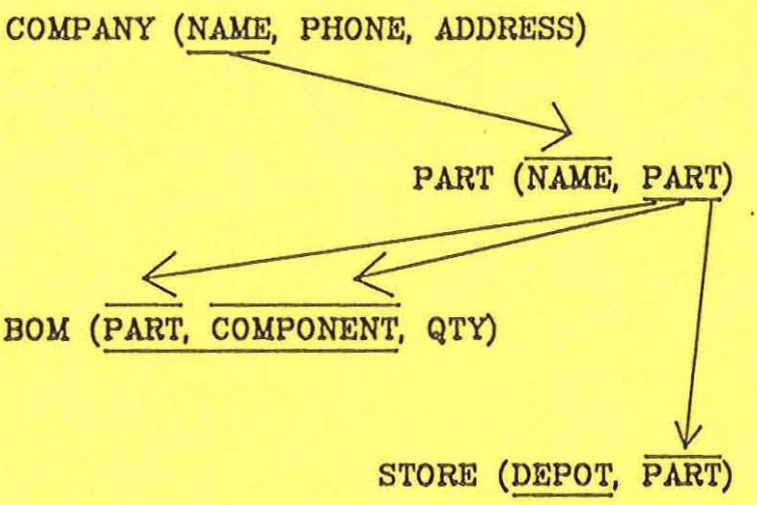

Figure 4

I N F O D I C GEMAP Information Dictionary (1.0)	elements	selection 1

element	type
ADDRESS	OBJ-LEX
ADDRESS	OBJ-LEX lexical object
BOM_ENTRY	ASS-DAT independent assoc.
BOM_QTY	ASS-DAT independent assoc.
BOOL	OBJ-LEX lexical object
COMPANY	OBJ-NLX nonlexical object
COMP_ADDR	ASS-DAT independent assoc.
COMP_PHONE	ASS-DAT independent assoc.
DATE	OBJ-LEX lexical object
DEPOT	OBJ-LEX lexical object
INT	OBJ-LEX lexical object
NAME	OBJ-LEX lexical object
NAM_CON__1	ASS-NAM naming convention
NAM_CON__2	ASS-NAM naming convention

Char Mode: Replace Page 1 Count: *0

Figure 5

I N F O D I C GEMAP Information Dictionary (1.0)	simple objects	selection 1

object PART_NO	type OBJ-LEX	
object NAME	type OBJ-LEX lexical object	
repres_by STRING	length 40	scale
object PART	type OBJ-NLX nonlexical object	
repres_by	length	scale
object PART_NO	type OBJ-LEX lexical object	
repres_by STRING	length 20	scale

^ Char Mode: Replace Page 1 Count: 9

Figure 6

I N F O D I C GEMAP Information Dictionary (1.0)	associations/ dependences	selection 2

association/dependence — type

BOM_ENTRY — ASS-DAT

BOM_ENTRY — ASS-DAT independent assoc.__
BOM_QTY — ASS-DAT independent assoc.__
COMP_ADDR — ASS-DAT independent assoc.__

role COMPONENT
object name PART — type OBJ-NLX
minimal occurence 0 — maximal occurence

role PART
object name PART — type OBJ-NLX
minimal occurence 0 — maximal occurence

Char Mode: Replace Page 1 Count: *0

Figure 7

I N F O D I C GEMAP Information Dictionary (1.0)	naming conventions	selection 2

naming convention NAM_CON__1

naming convention NAM_CON__1

nonlexical object	MIN	MAX	identifier	MIN	MAX
COMPANY	1	1	NAME	1	1

naming convention NAM_CON__2

nonlexical object	MIN	MAX	identifier	MIN	MAX
PART	1	1	PART_NO	1	1

automatic naming (y/n) _ naming convention

nlx. object — type MIN MAX
identifier — type MIN MAX

Char Mode: Replace Page 1 Count: 2

Figure 8