IDBD
AN INTERACTIVE DESIGN TOOL
FOR CODASYL-DBTG-TYPE DATA BASES

Roland Dahl[1]
Janis A. Bubenko jr[2]

The Systems Development Laboratory (SYSLAB)[3]

ABSTRACT

The Interactive Data Base Designer (IDBD) assumes as input a conceptual description of data to be stored in the data base (in terms of a binary data model) and an expected work-load in terms of navigations in the conceptual model. Extensive checking of input is performed. The designer has the possibility to restrict the solution space of the design algorithm by prescribing implementation strategies for parts of the binary model.

1. Chalmers University of Technology, S-41296 Gothenburg, Sweden

2. University of Stockholm, S-10691 Stockholm, Sweden

3. This work has been supported by the National Swedish Board for Technical Development

1. Introduction

Before we can design a data base schema, compatible with some existing Data Base Management System, we have to determine what kind of data it should contain and what kind of work-load, in terms of queries, updates, inserts and deletes it must be able to handle. In order to permit examination of alternative solutions the requirements must be stated in as implementation independent terms as possible. By 'implementation independent' we mean that there have been made no decisions on how to group data items in records, which access techniques to use and how to navigate in a structure of records and sets.

Designing a data base is thus only a (relatively small) part of a systems development process. It is preceded by a number of activities the purpose of which is to analyze corporate information needs and to specify the requirements of an information system to be developed.

The Interactive Data Base Designer (IDBD) presented in this paper has been developed to be compatible with two kinds of system design (philosophies) approaches.

The first kind, the analytical approach, proceeds through development phases like

- goal and problem analysis

- activity analysis etc.

and arrives at a comprehensive set of requirements specifications. This set also includes a conceptual information model of relevant parts of the enterprise and a set of information requirements [Bub-80]. The conceptual information model (CIM) describes and defines relevant phenomena (entities, relations, events, assumptions, inference rules etc.) of the Universe of Discourse (UoD). The CIM models the UoD in an extended time

perspective in order to capture dynamic rules and constraints.

The next step in this approach is to 'restrict' the CIM (from a time perspective point of view) and to decide what information to store in the data base and how to conceptually navigate in this set of information in order to satisfy stated information requirements (see Gus-82 for a comprehensive exposition of this problem).

If the information to be stored in the data base is defined in terms of a binary data model and the conceptual navigations are specified assuming such a model then this is the required input to IDBD.

The other approach to data base design is the experimental one. In this case we assume that a 'fast prototype' is developed by the use of the CS4 system [Ber-77A]. CS4 employs a binary data model and is thus compatible with IDBD. Experimental use of the prototype system can provide us with statistics of navigation types and frequencies.

It is, of course, also advantageous to use the experimental approach as a complement to the purely analytical one in order to avoid guesswork concerning the requirements and the work-load.

The DBTG-schema design algorithm of IDBD has the binary data model and a set of implementation strategies in common with design-aids developed at the University of Michigan [Mit-75, Ber-77B, Pur-79]. It differs, however, from them in several important respects

- the tool is interactive which gives the designer a possibility to monitor the design process
- comprehensive checking of the consistency of input data is performed (we have empirically found that it is difficult to supply a tool of this kind with correct input the first time; a waste of time and computer resources is the result of optimizing incorrect input)
- the designer has a possibility to prescribe certain implementation alternatives for parts of the model (or the whole model). This has the following advantages
  + the designer can test his/hers own solution alternatives which may be 'natural' or which have other, preferred, non-quantifyable properties
  + unnormal or 'non-sense' solutions can be avoided
  + the tool can be used to augment an existing DB-schema
  + the solution space can be drastically reduced thereby making IDBD a realistic tool also for design of large, complex data base schemata
- the description of the work-load is practically realistic as navigation in the conceptual binary model can be defined.

This paper describes and explains IDBD in terms of running a small sample case. Section 2 describes the input to IDBD - the conceptual binary data model and how to describe the work-load in terms of navigating in the model. User interaction, checking of input and how to supply IDBD with design directives is presented in section 3. The design algorithm and the results of performing a design run are given in section 4.

2. Input data

The input to the IDBD consists of the following types of data:

- description of the conceptual data model - its data item types (representing entity types) and relations
- a description of the workload of the model defined in terms of run-units
- a description of certain parameters to be considered by the design algorithm
- a set of design directives to the design algorithm restricting its solution space.

A consistency check is performed on the model and the work-load descriptions. Also an analysis is performed to estimate the number of references to the data items and the relations when navigating in the model.

The input, interaction and processing of IDBD will be illustrated by a small practical case, the enterprise GROSS. The following is assumed.

GROSS is a local supplier, i.e. it supplies parts to customers located in the same city. Parts are distributed by cars and one cargo is called a delivery. One delivery can contain several orders, 1 to 20. Every day there are several deliveries, 1 to 20. Customers send their orders to GROSS. An order includes 1 to 25 parttypes. When an order arrives, its day for delivery is determined.

The following information requirements - in terms of queries - are defined in the preceding design stages.

1. For a particular day, all the customers which are to be supplied, and for each customer: name and address.

2. For a particular parttype, the orders in which it is included and the day of delivery for each order.

3. For a particular parttype and for a particular customer, the total dollar amount for the part in order.

4. For all orders, print all parttypes with their amounts.

Assume also that the following transactions have been defined:

1. Deletion of a delivery.

2. Insertion of a delivery.

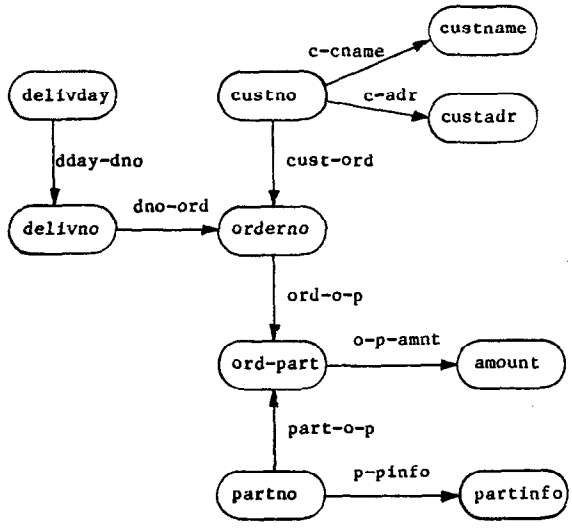
3. Insertion of a new customer.

4. Updating of part attributes.

This constitutes the basis for the conceptual (binary) data model and its work-load.

2.1 Description of the conceptual data model

We assume that the following binary relational data structure has been created on the basis of an analysis of the enterprise, the conceptual information model and the information requirements.



The model entity types are assumed represented in this case by a set of data item types. In fact, as "partinfo" in the data structure, a data item type can also represent a group of data item types.

Data item types are described with the word ITEM (in capital letters) on one line of input and thereafter, on separate lines, one per each data item:

- name of the data item
- size of the data item in number of characters
- cardinality of the data item
- a security code (optional)

If different data items cannot be placed in the same record, for instance because of security constraints or distributed data, then the data item types can be specified with different security code numbers. If the security code is not specified, it is put to zero.

In our case the data item types are specified as follows:

```
ITEM
delivno   5   150
delivday  6    30
orderno   6   300
ord-part  0  1500
partno    8  2000
partinfo 92  2000
amount    8  1500
custno    5  1000
custname 30  1000
custadr  30  1000
```

Binary relations in the data model are described with the word RELATION on one input line followed by one line per relation containing:

- name of the relation
- name of the data item from which the relation origins (item1)
- name of the data item to which the relation is directed (item2)
- the number of instances of the relation
- minimum number of item1 related to one item2
- maximum number of item1 related to one item2
- minimum number of item2 related to one item1

- maximum number of item2 related to one item1

The relations in our case are specified as follows:

RELATION							
dday-dno	delivday	delivno	150	1	1	1	20
dno-ord	delivno	orderno	300	1	1	1	20
cust-ord	custno	orderno	300	1	1	0	50
c-cname	custno	custname	1000	1	1	1	1
c-cadr	custno	custadr	1000	1	1	1	1
ord-o-p	orderno	ord-part	1500	1	1	1	25
o-p-amnt	ord-part	amount	1500	1	1	1	1
part-o-p	partno	ord-part	1500	1	1	0	200
p-pinfo	partno	partinfo	2000	1	1	1	1

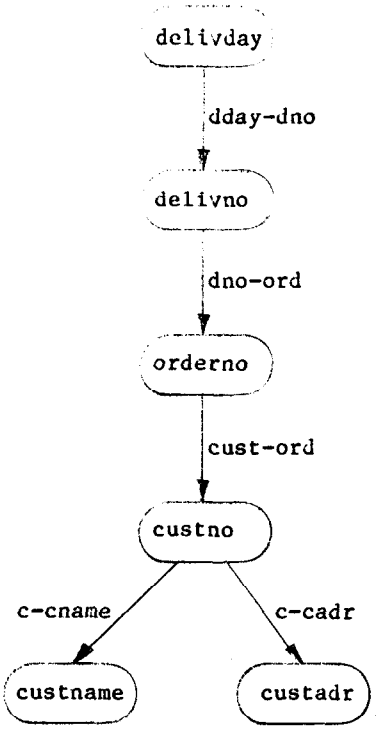
For the specified input data for a relation and the cardinality of the participated data items, IDBD determines the average number of item1(item2) related to item2(item1), the type of the relation (1:1, 1:M, M:1, M:N) and what type of mapping, total (T) and partial (P), that the domain and range of the relation participates in. This will be further discussed in section 3.1.

2.2 Description of the work-load

The work-load is generated by the information requirements (queries) and the transactions. They imply a need to **navigate** in the binary relational structure.

In order to show how navigations can be defined, two examples are given.

Example 1 For query 1 in our example the following access path is defined:



This access path could be described in natural language as follows:

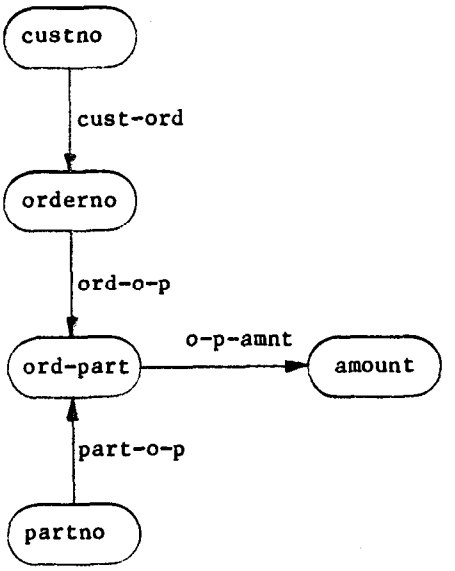
```
For a particular delivday
    get all delivno related to it via the relation
    dday-dno
        for all these delivno get all orderno
        which are related to all these delivno via
        the relation dno-ord
            for all these orderno get the custno
            related to these orderno via the
            relation cust-ord
                for the custno, get
                    custname related to it
                        via the relation c-cname,
                    custadr related to it
                        via the relation c-cadr
```

The navigation starts always at the level 1, where the entry-point, i.e. the starting item, and the operation for it is described. In example 1 the starting item is delivday.

After delivday has been accessed, all delivno related to delivday are to be accessed. This is done at the next higher level. We call the item delivday qualifier of delivno, because delivno related to delivday is requested. In the access path, each time when the latest accessed item is used as a qualifier for next item wanted, a next higher level is specified.

When the same item is used as qualifier for several required items, that can be specified at the same level. Custname and custadr have the same qualifier custno and therefore they are specified at the same level.

Example 2 In query 3 we want to get the dollar amount for the parts in order for a particular parttype and for a particular customer.



We can start the navigation either with access to partno or to custno. Here we choose to start with partno.

We access a unique partno, thereafter all ord-part related to it. At this stage we do not know for which of the ord-part we want to get the dollar amount, because we do not know to which customer ord-part is related. Therefore, we continue by accessing orderno for each of ord-part and then access the customer related to each order. Now we know which orders are related to the required customer. Now we need to know the ord-part related to those orders. We have already accessed ord-part and therefore we do not need to access them again. By a SELECT operation we can select the ord-part related to the required customer without additional accesses.

Going backwards in the access path can be done in two different ways. If the item to be accessed has the same qualifier as the item at the next higher level, then that higher level is specified. If one wants to skip one or more higher levels without making any access to the database, it can be done by using the special operation SELECT.

The following example shows the way query 1 and 3 can be expressed in terms of a run-unit.

```
RUN-UNIT
RU  custinf  30                    RU  amount  30
1                                  1
FIND UNIQ delivday                 FIND UNIQ partno
2                                  2
GET ALL delivno                    GET ALL ord-part
3                                  3
GET ALL orderno                    GET orderno
4                                  4
GET custno                         GET custno
5                                  2  3
GET custname                       SELECT ord-part
GET custadr                        3
                                   GET amount
```

IDBD uses the following syntactical rules for description of the run-units.

Run-units are described with the word RUN-UNIT followed by the run-units themselves. Every run-unit starts with a head line containing:

- the word RU
- name of the run-unit
- cardinality of the run-unit

After the head line the run-unit is described in a hierarchical way with level numbers much like a COBOL data declaration. On each line there is either a level description or a data operation description.

The level description lines are numbered from 1 and upwards. The first line after the head line is always a level description line with level number 1.

Each level description line contains:

- the level number
- optionally a cardinality

Normally the cardinality for a level is one. In that case there is no need to specify the cardinality number. Otherwise, a real number both < 1 (a probability) and > 1 (a frequency) can be specified. This cardinality multiplies with the cardinality on the next lower level to give the cardinality on the actual level. If a cardinality is specified on level 1, it multiplies with the cardinality of the run-unit.

On each level there can be specified zero, one or more occurrences of data operation descriptions. Every data operation description is defined on one line and contains:

- data operation verb
- name of the required data item
- optionally a relation name

Usually there is no need to specify a relation name. Only if there are more than one relation type between two data item types, it is necessary to specify the relation name. Otherwise the IDBD finds the relation type itself. The data item at the previous level is called the qualifier. At level 1, where there is no lower level, the access is directly to the data item type and not via a relation as at levels > 1.

The different data operation verbs are:

FIND [{UNIQ / SEQ}]

GET [{FIRST / LAST / NEXT / PRIOR / ITH / ALL}]

SELECT

{DELETE / INSERT / MODIFY} [{FIRST / LAST / ITH / ALL / UNIQ}]

{LINK / UNLINK} [ALL]

where { } means one of the elements inside { } and [ ] means that the element inside [ ] is optional.

FIND defines an entry point for the run-unit. FIND UNIQ implies some sort of direct access to the data item, while FIND SEQ implies a sequential browse through all data items of the mentioned data item type.

By GET one or all data items which are related to the qualifier are accessed.

By SELECT no access is made, because already accessed data items are selected. This normally also means a branch from a higher to some lower numbered level.

By DELETE one or all data items related to the qualifier are deleted. DELETE causes also deletion (i.e. UNLINK) of the relation instance, which has the deleted data item as its destination.

INSERT is analogous to DELETE.

By MODIFY one or all data items related to the qualifier are modified.

By LINK one or all data items are linked to the qualifier. This means that the relation instance(s) is(are) stored in the data base.

By UNLINK one or all data items are unlinked.

At level 1 only the following data operation verbs can be specified:

FIND [{UNIQ / SEQ}]

{DELETE / INSERT / MODIFY} [UNIQ]

A final example shows the definition of a run-unit describing the work-load of transaction nr 2 for inserting a new delivery. For the inserted delivno it is assumed that its delivday is not already stored in the data base with a probability of 20%.

```
RU  ins-supp  30
1
INSERT UNIQ delivno
2  0.2
INSERT delivday
2  0.8
LINK delivday
2
INSERT ALL orderno
3
LINK custno
INSERT ALL ord-part
4
INSERT amount
LINK partno
```

3. User interaction

3.1 Checking and analysis of input data

After the input phase and before the optimization phase, there is an interactive phase where also certain consistency checks are made.

First of all the IDBD asks for values of certain constants. These are

- maximum record length in number of characters

- counter length in number of characters

- pointer length in number of characters

- maximum secondary storage in number of characters that is allowed

- CALC storage factor, a factor greater or equal to one, which tells how much more secondary storage than nominal is required for hashed storage

- CALC access factor, a factor greater or equal to one, which tells how many more accesses than one that is required due to synonyms in hashed storage.

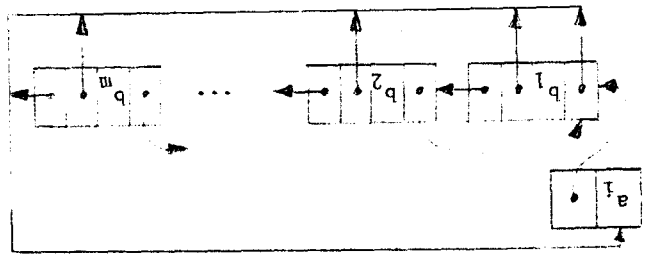
From the values of the first three of these constants and from the earlier description of data items, relations and run-units, the program decides which consistency constraints must hold. IDBD finds out for each relation which implementation alternatives (IAs) are **possible** , which are **unwanted** and which IAs are **impossible** (OFF). The lists of IAs for these three cases can be displayed for the DBA. For the possible and unwanted cases the DBA can interactively change IAs. There is also a fourth case that the DBA can use, namely to specify that one special IA is to be used (ON).

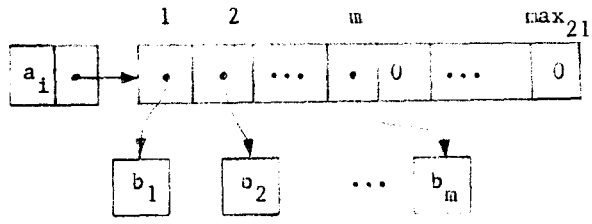
The reason for changing the lists of IAs can be that an IA is already fixed or that the IA gives an unnormal solution. (An example of an unnormal solution would be the case, where "article-number" is suggested to be aggregated under "number-in-stock" instead of the other way around.) If the DBA reduces the solution space for the different relations, the execution time during the optimization phase is also substantially reduced. This is necessary for a large data base.

There are 21 different implementation alternatives, numbered 1-21 (see also [Ber-77B]). The IAs are displayed with their number. Here is a short description of each IA:

1 = Fixed duplication
2 = Fixed duplication reversed
3 = Variable duplication
4 = Variable duplication reversed
5 = Fixed aggregation
6 = Fixed aggregation reversed
7 = Variable aggregation
8 = Variable aggregation reversed
9 = Chain, next pointer
10 = Chain, next pointer reversed
11 = Chain, next + owner pointer
12 = Chain, next + owner pointer reversed
13 = Chain, next + prior pointer
14 = Chain, next + prior pointer reversed
15 = Chain, next + owner + prior pointer
16 = Chain, next + owner + prior pointer rev.
17 = Pointer array
18 = Pointer array reversed
19 = Pointer array + owner pointer
20 = Pointer array + owner pointer reversed
21 = Dummy record

For illustration, the implementation alternatives 15 and 17 are shown below.





After the user has finished the change of the IAs, (see section 3.2) IDBD takes the best remaining IAs for every relation, e.g. it takes the set of IAs, which are either ON, possible or unwanted, in that order. The number of possible alternatives are multiplied and the total number of combinations of DBTG-structures are displayed for the user. The user has now the possibility to further reduce the solution space. The reason for this is that the computing time can be very long, if there are many DBTG-structure alternatives to consider. It is also possible to limit the processing time to some value. After that time the optimizing phase is interrupted and the next phase in the program continues.

For the different valid DBTG-structures IDBD calculates an estimate of the number of accesses to the data base for every run-unit. The need for secondary storage is also calculated.

The best solutions are presented to the user. "The best solutions" are those with the least number of accesses for a certain secondary storage size. The solutions are presented in the order of increasing number of accesses. At the same time the secondary storage requirements are decreasing in order to belong to the set of best solutions. (The first 10 solutions are always added to the set.)

For different DBTG-structures IDBD also gives some messages such as

- an entry point for an item is missing
- there is no access path to an item
- there is no SYSTEM entry (sequential access) to an item

A typical user interaction is examplified in the next section.

The analysis of input data will be illustrated by the following examples. IDBD has given the following output from the input data checking procedures for this particular case:

Analysis of data items:

Data items

name	size	card.	DA-ref.no	seq.ref.no	secur.
delivno	5	150	120		
delivday	6	30	30		
orderno	6	300		300	
ord-part	0	1500			
partno	8	2000	1630		
partinfo	92	2000			
amount	8	1500			
custno	5	1000	20		
custname	30	1000			
custadr	30	1000			

Explanation:

- DA-ref.no means the number of references (logical accesses), which are needed directly to the data item type
- seq.ref.no means the number of sequential accesses, which are needed to the data item type.

These numbers give the data base administrator or designer some indication of to which data items there will be a need for direct access and/or sequential access.

Analysis of relations:

Relations

rel.name	type	T/P	item1	item2	card.	min/av/max12			min/av/max21		
dday-dno	1:M	TT	delivday	delivno	150	1	1.0	1	1	5.0	20
dno-ord	1:M	TT	delivno	orderno	300	1	1.0	1	1	2.0	20
cust-ord	1:M	PT	custno	orderno	300	1	1.0	1	0	0.3	50
c-cname	1:1	TT	custno	custname	1000	1	1.0	1	1	1.0	1
c-cadr	1:1	TT	custno	custadr	1000	1	1.0	1	1	1.0	1
ord-o-p	1:M	TT	orderno	ord-part	1500	1	1.0	1	1	5.0	25
o-p-amnt	1:1	TT	ord-part	amount	1500	1	1.0	1	1	1.0	1
part-o-p	1:M	PT	partno	ord-part	1500	1	1.0	1	0	0.8	200
p-pinfo	1:1	TT	partno	partinfo	2000	1	1.0	1	1	1.0	1

Explanation:

- type is the type of mapping (1:1, 1:M, M:1, M:N)
- T/P stands for total (T) and partial (P) mappings between the data items in the domain and range of the relation
- min/av/max12 shows the minimum, average and maximum number of item1 related to item2
- min/av/max21 is analogous, but concerns the reverse relation

Analysis of reference frequencies to relations:

Relations with reference count

rel.name	item1	item2	ref.no fwd number	ref.no fwd %	ref.no bwd number	ref.no bwd %
dday-dno	delivday	delivno	150	1.2	141	1.1
dno-ord	delivno	orderno	540	4.2	75	0.6
cust-ord	custno	orderno			443	3.5
c-cname	custno	custname	320	2.5		
c-cadr	custno	custadr	320	2.5		
ord-o-p	orderno	ord-part	2700	21.1	98	0.8
o-p-amnt	ord-part	amount	2790	21.8		
part-o-p	partno	ord-part	98	0.8	2100	16.4
p-pinfo	partno	partinfo	3000	23.5		

Explanation:

- rel.no fwd shows the number of required references from item1 to item2 in the relation (forwards). Every update operation (DELETE, INSERT and MODIFY) is calculated as 2 references.
- rel.no bwd shows the number of required references from item2 to item1 in the relation (backwards).

These numbers give the designer some indication of which relations are critical for the efficiency of the data base system. Those relations can then be analyzed in greater detail.

The analysis of run-units shows for each run-unit the following type of output (examplified for run-units "custinf", "amount" and "ins-supp").

```
Run-units
ru-name  cardinality
  level-no  cardinality
    operation    item1   item2  relation  reversed

custinf    30.00
  1
    FIND  UNIQ  delivday
    2
      GET   ALL   delivno  delivday dday-dno
      3
        GET   ALL   orderno  delivno  dno-ord
        4
          GET         custno   orderno  cust-ord rev.
          5
            GET         custname custno   c-cname
            GET         custadr  custno   c-cadr

amount     30.00
  1
    FIND  UNIQ  partno
    2
      GET   ALL   ord-part partno   part-o-p
      3
        GET         orderno  ord-part ord-o-p  rev.
        4
          GET         custno   orderno  cust-ord rev.
    2     3.00
      SELECT      ord-part
      3
        GET         amount   ord-part o-p-amnt

ins-supp   30.00
  1
    INSERT UNIQ delivno
    2     0.20
      INSERT      delivday delivno  dday-dno rev.
    2     0.80
      LINK        delivday delivno  dday-dno rev.
    2
      INSERT ALL  orderno  delivno  dno-ord
      3
        LINK        custno   orderno  cust-ord rev.
        INSERT ALL  ord-part orderno  ord-o-p
        4
          INSERT      amount   ord-part o-p-amnt
          LINK        partno   ord-part part-o-p rev.
```

Explanation:

The "Run-units" listing is merely just a structured reprint of the input data with the relations expanded and a note "rev." if the relation is used in backward direction, i.e. from item2 to item1.

3.2 Design directives

Supplying design directives to the IDBD for choosing among implementation alternatives is best illustrated by showing part of a typical interactive user-IDBD session.

The following is an example of a user interaction with the IDBD:

```
Submit values to the following constants
maximum record length=512
counter length=2
pointer length=4
max. secondary storage=100000000
CALC storage factor=1.2
CALC access factor=1.5
Look at/change implementation alternatives? (y/n)y

Instructions? (y/n)y
```

You will see all or specified relations with their relation name and what consistency constraints that hold for the different implementation alternatives.

```
The different consistency constraints are:
ON         = the relation shall have this impl. alt.
POSSIBLE   = possible impl. alternatives
UNWANTED   = not desirable impl. alternatives
OFF        = these impl. alt. are not permitted

You can change among the ON, POSSIBLE and
UNWANTED implementation alternatives by writ-
ing the implementation alternative number and
the letter O, P or U respectively on one line

Do you want all or specified relations
displayed? (s/s)a

dday-dno
ON=
POSSIBLE=  1  2  3  5  7 11 15 19
UNWANTED=  9 13 17
OFF=       4  6  8 10 12 14 16 18 20 21
change?y
change=15 o
more changes?n
dday-dno
ON=       15
POSSIBLE=  1  2  3  5  7 11 19
UNWANTED=  9 13 17
OFF=       4  6  8 10 12 14 16 18 20 21
dno-ord
ON=
POSSIBLE=  1  2  3  5  7 11 15 19
UNWANTED=  9 13 17
OFF=       4  6  8 10 12 14 16 18 20 21

- - - - - -

p-pinfo
ON=
POSSIBLE=  5
UNWANTED=  6 9 10 11 12
OFF=       1 2 3 4 7 8 13 14 15 16 17 18 19 20 21
change?n

Note: For all 1:M-relations except for dno-
ord the suggested IA is in this example put
to 15, e.g. chain with both owner and prior
pointer. For all 1:1-relations the only pos-
sible IA is left with number 5, e.g fixed
aggregation.

IDBD will then continue with the following
listing:

There are 8 different DBTG-structures to examine.
Do you want to reduce the solution space further?n
Do you want to limit the CPU-time when optimizing?n

Note: Normally IDBD will save the best 10
solutions (with least number of accesses to
the data base). In this case there are only
8 possible solutions, of which 6 of these are
accepted as correct DBTG-structures.
```

4. The Design-aid

4.1 The algorithm

4.1.1 **Interactivity** The three different design tools developed at the University of Michigan [Mit-75, Ber-77B, Pur-79] all work in a similar way. The design aids are typically batch programs. From a specified input the tools produce, after an optimization phase, efficient data structures. Some problems with this type of tools are that

- they sometimes produce unnormal solutions
- they restrict themselves to a reduced solution space
- the designer has no possibility of testing his/her own data structures, which may be more natural or which may have other (non-quantifiable) desirable properties
- they take, in spite of sophisticated optimization algorithms, too long time to run on a computer for normal sized data bases.

To overcome these problems the design aid has to be interactive. The data base designer has then the possibility to manipulate with the solution space so that these problems do not have to arise.

4.1.2 **Validity constraints** Certain validity constraints must be satisfied in order to arrive at a valid DBTG data structure.

Some of these rules cannot be checked until a DBTG structure with records and sets is created. The validity constraints, which are then tested, are

- record lengths are within permitted limits
- a record type has no repeating groups on a level > 1
- a data item type is not aggregated more than once
- that a set has not the same record type both as an owner and as a member.

Most of the validity constraints can, however, be checked before the interaction with the data base designer takes place. This is done by determining which IAs are valid or not valid for every relation. Unlike the Michigan design tools, IDBD does not just distinguish between valid and not valid IAs but also categorizes valid IAs into preferable(possible) and unwanted IAs. In this way the tool helps the designer to choose good data structures. If the designer lets IDBD to decide, IDBD has then a smaller number of IAs to consider. The solution space is thereby considerably reduced.

Some of the validity constraints, which are checked before the interaction, are

- maximum record length is not violated
- if the data item types in the relation have different security codes, duplication and aggregation are ruled out
- for a relation, where there exists a partial mapping from A to B, such that there exist B data items that are not related to any A data item, it is impossible to aggregate B under A such that all B data items are represented.

Suppose there is a 1:M-type of relation between A and B data items so that one A data item is related to zero, one or more B data items. The validity constraints, which are checked in this case, are (there are analogous rules for 1:1- M:1- and M:N-relations)

- it is impossible to aggregate A under B
- a chain or pointer array can not be used in reversed direction
- there is no need for dummy records
- duplicating of A under B is done with fixed length, not variable length
- if references in the run-units only traverse in the direction of the relation, e.g. from A to B, there is no need for owner pointers or for duplication of A under B
- if there is only traversing in the opposite direction, duplication or aggregation of B under A and chain or pointer array without owner pointer are made unwanted
- if there is traversing in both directions, we need owner pointers, e.g. chain or pointer array without owner pointer are made unwanted.

4.1.3 **Cost calculation** The storage cost is calculated as the sum of the lengths of all records, pointers and wasted areas for hashed record types. These wasted areas are estimated by the aid of the parameter "CALC storage factor".

There are a few assumptions about the record type implementation and the DBTG data base management system. If a record type is only accessed with direct access, then it is assumed that the record type will be hashed (CALC in Codasyl). If a record type is accessed sequentially, then the record type is made a member in a Codasyl SYSTEM set (Singular set). If there is a need for direct access to more than one data item type in a record type, then it is created a secondary index. This index is assumed to be implemented by a pointer array.

The access cost is calculated for each run-unit. We differentiate between three types of access costs.

These are the number of

- sequential accesses, where a scan through the records for a certain record type is made. These accesses can be cheaper than the other types, if the records are clustered into blocks.
- CALC accesses, which are accesses to hashed records. The number is multiplied with the parameter "CALC access factor"
- pointer accesses, which are the accesses through pointers.

The number of accesses calculated is an estimation of the number of physical accesses. Records, which are stored in the same block or already stored in primary storage in buffers, are not considered.

To get an estimate of the number of accesses, IDBD takes care of the hierarchical structure in the run-units with different number of data items required on each level. Average values are used. IDBD also considers

- the cases where the data item types are stored in the same record type
- what kind of data entry there is defined to the record type
- the combination of IAs and verbs in the run-unit.

As an example of the number of pointer accesses calculated, consider the verbs GET FIRST, MODIFY FIRST and INSERT FIRST for an implementation alternative with a chain with prior pointers, for instance IA=15. Table 1 shows the figures.

Run-unit verb	Number of pointer accesses
GET FIRST	1
MODIFY FIRST	2
INSERT FIRST	5

TABLE 1. Number of pointer accesses for certain verbs

In the case of INSERT FIRST both the owner and the previous first member have to be read and written and the new first member shall be written, e.g. 5 accesses.

4.2 **The results**

Each solution alternative contains information about

- record structures
- set types
- total storage cost
- total number of accesses and
- the access cost for each run-unit.

This information is, for each solution, displayed as follows (examplified by the solution alternative with the lowest access cost).

```
There are 6 record types

Record type no  1 (delivday) has    30 records
 0 delivday       6 char. CALC access
Record length:      6 char.

Record type no  2 (delivno ) has   150 records
 0 delivno        5 char. CALC access
Record length:      5 char.

Record type no  3 (orderno ) has   300 records
 0 orderno        6 char. seq. access
Record length:      6 char.

Record type no  4 (custno  ) has  1000 records
 0 custno         5 char. CALC access
   1 custname    30 char.
   1 custadr     30 char.
Record length:     65 char.

Record type no  5 (ord-part) has  1500 records
 0 ord-part       0 char.
   1 amount       8 char.
Record length:      8 char.

Record type no  6 (partno  ) has  2000 records
 0 partno         8 char. CALC access
   1 partinfo    92 char.
Record length:    100 char.
```

```
There are  5 set types

Set type no  1 (dday-dno) has     150 instances
Owner  = record type no  1         30 records
Member = record type no  2        150 records
The set is implemented with DBTG-set
with owner pointer and prior pointer

Set type no  2 (dno-ord ) has     300 instances
Owner  = record type no  2        150 records
Member = record type no  3        300 records
The set is implemented with DBTG-set
with owner pointer

Set type no  3 (cust-ord) has     300 instances
Owner  = record type no  4       1000 records
Member = record type no  3        300 records
The set is implemented with DBTG-set
with owner pointer and prior pointer

Set type no  4 (ord-o-p ) has    1500 instances
Owner  = record type no  3        300 records
Member = record type no  5       1500 records
The set is implemented with DBTG-set
with owner pointer and prior pointer

Set type no  5 (part-o-p) has    1500 instances
Owner  = record type no  6       2000 records
Member = record type no  5       1500 records
The set is implemented with DBTG-set
with owner pointer and prior pointer

  The storage cost for the data base is 391836 char.

  Run-unit custinf  has the access cost:   30 times
    seq. access   =    0
    CALC access   =    1
    pointer access=   25

  Run-unit ord-day  has the access cost:  100 times
    seq. access   =    0
    CALC access   =    1
    pointer access=    3

  Run-unit amount   has the access cost:   30 times
    seq. access   =    0
    CALC access   =    1
    pointer access=    2

  Run-unit orders   has the access cost:    1 times
    seq. access   =  300
    CALC access   =    0
    pointer access= 3000

  Run-unit del-supp has the access cost:   30 times
    seq. access   =    0
    CALC access   =    1
    pointer access=   77

  Run-unit ins-supp has the access cost:   30 times
    seq. access   =    0
    CALC access   =    1
    pointer access=  106
  Run-unit ins-cust has the access cost:   10 times
    seq. access   =    0
    CALC access   =    1
    pointer access=    3

  Run-unit upd-part has the access cost: 1500 times
    seq. access   =    0
    CALC access   =    1
    pointer access=    1

  The total number of accesses=  14045
```

Table 2 shows the storage and access costs for the 6 alternatives in this sample case.

Solution number	Access cost	Storage cost	No. of record types	No. of set types
1	14045	391836	6	5
2	14165	393036	6	5
3	14315	420636	6	5
4	76512	390336	6	4
5	100025	391356	6	4
6	100025	410436	6	4

TABLE 2. Access and storage costs

The schemata for solution alternatives 1 and 4 are graphically illustrated in figures 1 and 2.

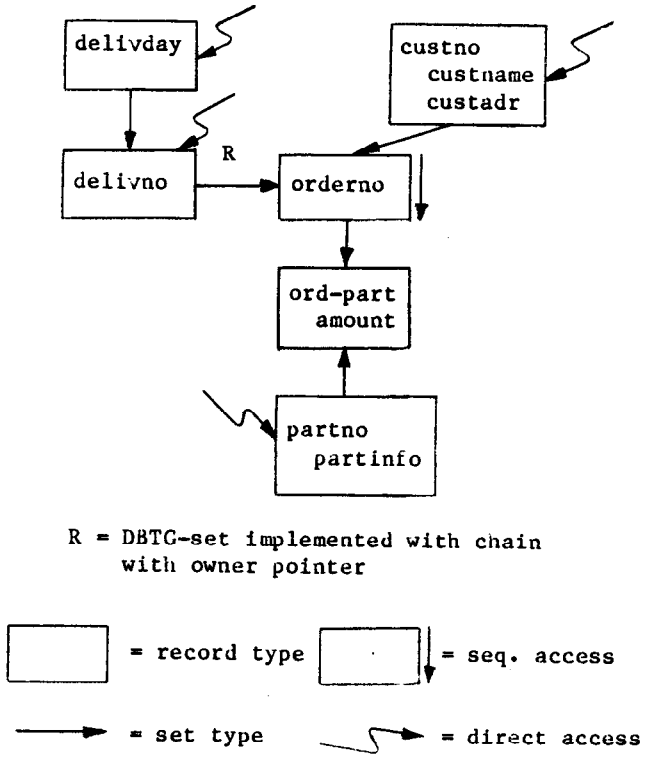


Figure 1. Solution alternative no. 1

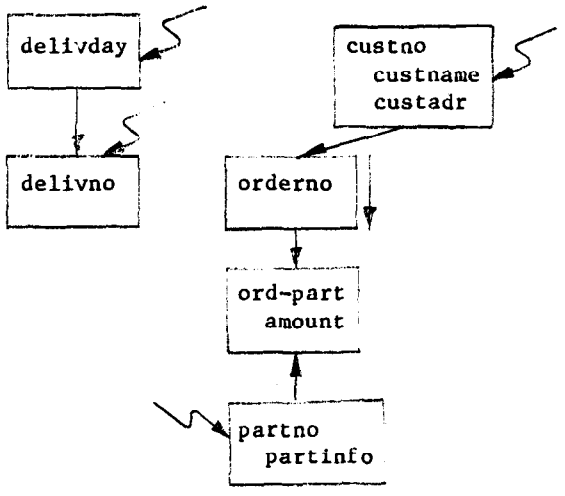


Figure 2. Solution alternative no. 4

In solution alternative no. 4 is delivno duplicated under orderno. This means that to get an orderno from a given delivno in the records with orderno+delivno, those records have to be scanned. That is why the access cost in this case raises tremendously.

4.3 The tool

IDBD is running on VAX computers with UNIX operating system. The program call is a standard UNIX call:

idbd [-i infile] [-o outfile] [-p parmfile] [-r rfile] [-s sofile]
[-u sifile]

It is possible to name the different files at the program call with parameters. Otherwise the names of the files will be asked for by the IDBD during execution. The different files are

infile — The name of the input data file, which contains the data items, the relations and the run-units.

outfile — The name of the file, where the lengthy listings are stored. The filename /dev/tty means the terminal itself. The filename /dev/null produces no output file.

parmfile — Instead of answering questions about the values for the different parameters, these can be stored in a file. The name of that file is given with this parameter.

rfile — If there is just one DBTG structure to analyze, put the IAs for the relations 1, 2 etc. in a file and name that file with this parameter at the program call.

sofile — It is possible to interrupt the program after the input phase in order to examine the output from the input checking procedures. Give in that case this parameter. The data is saved in the file sofile.

sifile — To continue processing after an interrupt with -s sofile give the name of the saved file with this parameter.

5. Discussion

Discussions with practitioners in the field has disclosed that one is not always interested in 'optimizing' the total DB-schema. For many reasons (reliability, security, modifyability, comprehensibility etc.) the designers may wish to implement parts of the data base in a specific way and leave the rest of it (if any) to the 'optimizer'. The difficulty of predicting future workloads may also make optimization – in the strict sense – more an intellectual exercise than a realistic approach. An optimal solution alternative may, nevertheless, have a merit. It provides a 'base-line' against which other, more 'natural', solutions can be measured in terms of access and storage costs.

IDBD has the flexibility to optimize the design from a predefined, desired scope. Tests show that it is a valuable property in order to make a tool useful in a variety of practical design situations.

REFERENCES

[Ber-77A] Berild S., Nachmens S.: A Tool for Data Base Design by Infological Simulation, Proc. 3rd Int. Conf. on VLDB, Tokyo, 1977.

[Ber-77B] Berelian E.: A Methodology for Data Base Design in a Paging Environment, The University of Michigan, Systems Engineering Laboratory, Ann Arbor, USA, Ph.D. Thesis, 1977.

[Bub-80] Bubenko jr J.A.: Information Modeling in the Context of Systems Development, in S.H. Lavington (Ed.) "Information Processing 80", North Holland, 1980, pp. 395-411.

[Gus-82] Gustafsson M.R., Karlsson T., Bubenko jr J.A.: A Declarative Approach to Conceptual Information Modeling, IFIP WG8.1 Working Conference "Comparative Review of Information System Design Metho-

dologies", North Holland, 1982.

[Mit-75] Mitoma M.F., Irani K.B.: Automatic Database Schema Design and Optimization, Proc. of 1st Int. Conf. on VLDB, 1975.

[Pur-79] Purkayastha S.: Design of DBMS-processable Logical Database Structures, The University of Michigan, Ann Arbor, USA, Ph.D. Thesis, 1979.